BRUNO GUEDES SOUTO

TRANSMISSÃO DE DADOS ESTEGANOGRAFADOS E CRIPTOGRAFADOS NO CABEÇALHO DE PACOTES TCP/IP

Trabalho de conclusão de curso apresentado à banca examinadora do Centro Universitário de Brasília, como exigência parcial para conclusão do curso de Engenharia de Computação, sob orientação do Professor MsC. Francisco Javier De Obaldía Díaz.

Brasília – Distrito Federal
Dezembro/2008

Aos meus pais que me deram a educação e formação necessária para hoje ocupar este lugar, a todos os amigos e professores que me apoiaram, e não deixaram eu desistir mesmo nos momentos mais difíceis da realização deste trabalho.

RESUMO

Este projeto consiste em desenvolver um método que permita trafegar dados sigilosos numa rede utilizando tecnologia TCP/IP sem que qualquer pessoa que a esteja monitorando perceba o tráfego de dados sigilosos. Para tal, a idéia é desenvolver uma aplicação que utilize um método de esteganografia aliado a um algoritmo de criptografia para enviar dados sigilosos pela rede.

A implementação consiste em criptografar uma entrada de dados utilizando o algoritmo Ron Rivest, Adi Shamir e Len Adleman (RSA) e, então, mascará-la de modo esteganográfico, no cabeçalho TCP/IP, para assim enviá-la por uma rede de dados com o objetivo de ser recuperada por outra máquina onde será decifrada. São apresentados como resultados do trabalho o programa escrito para realizar essas operações e os testes realizados.

Palavras-chaves: esteganografia, criptografia, RSA, segurança, redes, Internet.

ABSTRACT

This dissertation’s objective is to develop a method to traffic secretive data over a network using TCP/IP technology without any third party that monitors this traffic be able to acknowledge it. For in such a way, the main idea is to develop a application that uses a steganography method allied to cryptographic algorithm to send those data over the network.

The implementation consists of ciphering a string of text using the RSA algorithm and then hide it in a steganographic way inside the TCP/IP header in order to send it over a network to another machine to decipher it. The program to do such and the tests performed are presented as results of this project.

Keywords: steganography, criptography, RSA, security, networks, Internet.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

RSA - Algoritmo Ron Rivest, Adi Shamir e Len Adleman

TCP - Transport Layer Protocol

IP - Internet Protocol

ISN - Campo do Número Seqüencial

PAR - *Positive Acknowledgement with Retransmission*

API - Application Programming Interface

SSL - Secure Sockets Layer

TLS - Transport Layer Security

LISTA DE SÍMBOLOS

Bits – Dígito binário

Mbits – Mega bits

KB/seg – Quilobytes por segundo

KB - Quilobytes

GB - Gigabytes

1. CAPÍTULO 1 – INTRODUÇÃO

Este trabalho trata dos métodos de criptografia e esteganografia para proteger informação aliados a transmissão de rede utilizando o protocolo TCP/IP.

Nesta introdução serão tratados os motivos, objetivos e a estrutura em que este projeto esta fundamentado.

1.1. Motivação

A esteganografia é a arte e ciência de escrever mensagens escondidas de uma maneira que ninguém além do remetente e do receptor da mensagem consiga sequer perceber que exista uma mensagem escondida.

Em contraste com a esteganografia temos a criptografia, a criptografia não esconde o fato que existe uma mensagem sendo transmitida/enviada, ela apenas impede a leitura da mensagem de alguma maneira. (WAYNER, Peter, Disappearing cryptography: information hiding: steganography & watermarking, 2002).

Este trabalho foi motivado após uma vasta pesquisa na internet por programas e técnicas de esteganografia, fazendo essa pesquisa houve a percepção de que as técnicas de esteganografia mais em voga hoje são as técnicas que utilizam imagens ou arquivos de áudio para esconder a mensagem desejada, assim como os softwares disponíveis livremente também. Com isso, houve a idéia de utilizar outra técnica bastante falada nos livros de segurança mas pouco implementada, que é a de esconder a informação dentro de alguma parte da camada de transporte. Com essa idéia em mente selecionamos para este trabalho esconder a informação utilizando o TCP (Transport Layer Protocol) e o campo do número seqüencial (ISN). Para tornar a informação mais segura ainda será utilizado um método de criptografia assimétrico para criptografar a mensagem antes desta ser escondida na camada de transporte.

1.2. Objetivo

1.2.1. Objetivo Geral

Provar a possibilidade de esteganografar pequenas mensagens na camada de transporte utilizando um campo específico do cabeçalho TCP.

1.2.2. Objetivo específico

Criar uma ferramenta capaz de criptografar, esteganografar e enviar pequenas mensagens pela rede utilizando para isso o protocolo TCP/IP. Esteganografando a mensagem dentro do cabeçalho TCP e a criptografando utilizando um algoritmo de criptografia assimétrico. Além disso, estudar os impactos da transmissão desta mensagem na rede e estudar a confiabilidade e confidencialidade desta mensagem quando transmitida.

1.3. Estrutura da monografia

Este projeto se estrutura em sete capítulos principais e suas subdivisões em tópicos quando necessário a descrição mais detalhada de cada tema.

O capítulo um trata de forma introdutória do que se trata este trabalho. Abordando a motivação, os objetivos, resultados esperados e a estrutura da monografia.

No capitulo dois são apresentados os conceitos de rede, conexões orientadas e não orientadas a conexão e um aprofundamento nos protocolos TCP

e IP apresentando os conceitos e referencial teóricos que são utilizados no desenvolvimento deste trabalho

No capítulo três finalizamos o referencial teórico apresentando os conceitos de esteganografia, criptografia simétrica e assimétrica e uma explicação geral do algoritmo RSA.

O capítulo quatro apresenta uma visão geral do projeto e apresenta as funcionalidades implementadas no software desenvolvido.

No capítulo cinco tratamos das principais funções de código utilizadas para desenvolver o software, explicando as funcionalidades implementadas e como foram implementadas.

O capítulo seis traz os resultados dos testes realizados com a ferramenta, sendo os testes de rede, confiabilidade e confidencialidade.

E finalmente no capítulo sete temos a conclusão do trabalho, com as dificuldades enfrentadas e sugestões para projetos futuros.

1.4. Resultados Esperados

O resultado esperado deste trabalho é um software capaz de criptografar, esteganografar e enviar uma pequena mensagem pela rede entre dois computadores garantindo a confiabilidade e confidencialidade mesmo quando o tráfego de rede estiver sendo capturado ou espionado por um usuário mal intencionado.

2. CAPÍTULO 2 - REDES TCP/IP E INTERNET

Desde a década de 60 a tecnologia de redes vem sendo observada e pesquisada pelo governo dos Estados Unidos, assim como por outros governos interessados em utilizar serviços associados. A importância e o potencial dessas redes hoje todos conhecemos como a Internet Global (COMER, Douglas, Interligação de redes com TCP/IP, 2006). Essa tecnologia é fruto das idéias e princípios que foram resultado da pesquisa financiada pelo Defense Advanced Research Projects Agency (DARPA) Esta pesquisa ajudou a criar “um conjunto de padrões de rede que especifica os detalhes de como os computadores se comunicam, além de um conjunto de convenções para interconexão de redes e encaminhamento de tráfego” (COMER, 2006, p. 2). Este conjunto de convenções deu origem ao oficialmente chamado hoje de TCP/IP Internet Protocol Suíte, comumente encurtada apenas para TCP/IP, que seriam os nomes dos seus dois padrões principais:

- *Transport Control Protocol* (Protocolo de Controle de Transporte) e
- *Internet Protocol* (Protocolo Internet)

A tecnologia TCP/IP permite a comunicação entre qualquer conjunto de redes interconectadas. Por exemplo, a rede dentro de um apartamento com 2

computadores, as redes de um prédio, as redes de uma universidade e até várias redes espalhadas pelo mundo. Devido à viabilidade demonstrada em grande escala o padrão TCP/IP se tornou o coração da Internet Global e hoje conecta mais de 650 milhões de pessoas espalhadas por todo o mundo (COMER, Douglas, Interligação de redes com TCP/IP, 2006).

2.1. Comunicação em Redes

Existem dois tipos de redes de comunicação básicas que podemos citar, as orientadas a conexão e as sem conexão.

2.1.1. Redes orientadas a conexão

As redes orientadas a conexão se comportam formando uma conexão dedicada entre si, comumente chamada de circuito. O exemplo mais comum que podemos oferecer de uma rede de conexão seria o de uma chamada telefônica tradicional. “Uma chamada telefônica estabelece uma conexão do telefone de origem através da central de comutação local, através de linhas tronco, até uma central de comutação remota, e finalmente ao telefone destino.” (COMER, Douglas, Interligação de redes com TCP/IP, 2006)

A vantagem das redes orientadas a conexão está na sua garantia de capacidade, ou seja, nenhum tipo de atividade na rede irá influenciar na capacidade da conexão depois que um circuito for estabelecido. (COMER, Douglas, Interligação de redes com TCP/IP, 2006) No exemplo da ligação telefônica existe um caminho de dados garantido de 64 Kbps que seria a taxa necessária para o tráfego de voz digitalizada.

A desvantagem das conexões orientadas a conexão está no fato de que ela é um tipo de comunicação mais custoso, visto que não importa se a rede está sendo usada ou não já que os custos dos circuitos são fixos. (COMER, Douglas, Interligação de redes com TCP/IP, 2006) Logo, numa ligação telefônica não importa se as duas pessoas estão conversando ou caladas, ainda existe um custo fixo por minuto naquela ligação.

2.1.2. Redes sem conexão

As redes sem conexão funcionam de uma maneira bem diferente das redes orientadas a conexão. Primeiramente, elas fracionam os dados a serem transferidos em unidades bem menores, de apenas algumas centenas de bytes. Essas pequenas unidades são chamadas de pacotes, esses pacotes são então reunidos e transmitidos (multiplexados) por conexões entre máquinas de alta

capacidade. Esses pacotes transportam, alem de um pedaço dos dados do arquivo original, várias informações de identificação. Essas informações são o que permitem que o hardware de rede saiba para onde os pacotes vão e que o software os remonte num único arquivo novamente em seu destino (COMER, Douglas, Interligação de redes com TCP/IP, 2006).

A vantagem das redes sem conexão é que podem ocorrer várias comunicações concorrentes entre os computadores utilizando “conexões compartilhadas entre máquinas por todos os pares de computadores que estão se comunicando”. (COMER, Douglas, Interligação de redes com TCP/IP, 2006)

A desvantagem está no fato de que se a rede ficar sobrecarregada algum par de computadores irá ter que funcionar abaixo da capacidade da rede, logo tendo que esperar para mandar mais pacotes.

Diferente das redes orientadas a conexão, as redes sem conexão não são capazes de garantir a capacidade da rede, mesmo assim a sua popularidade é crescente devido ao seu baixo custo, como menos conexões são necessárias porque várias máquinas podem compartilhar a largura de banda da rede, mantendo assim o custo baixo. (COMER, Douglas, Interligação de redes com TCP/IP, 2006). E como hoje, com a engenharia avançada, temos hardwares de

rede capazes de suportar grandes velocidades de transferência temos que a capacidade não chega a se tornar um problema. Para referência no restante desta monografia, quando for utilizado o termo rede sempre estaremos considerando uma rede sem conexão.

2.2. Internet Protocol – IP

O IP é um protocolo de entrega de datagrama sem conexão (COMER, Douglas, Interligação de redes com TCP/IP, 2006). Um datagrama consiste da unidade de transferência básica da Internet e é dividido em áreas de cabeçalho e dados. O cabeçalho contém informações como, por exemplo, o endereço de origem e destino do datagrama e a área de dados contêm os dados transportados pelo datagrama.

2.2.1. Arquitetura e Filosofia da Internet

Para entendermos como funciona a internet é comum utilizarmos um modelo conceitual de 3 camadas que representam cada uma um conjunto de serviços (COMER, Douglas, Interligação de redes com TCP/IP, 2006). Essa separação conceitual ajuda a indicarmos os detalhes filosóficos do projeto e traz a

vantagem de podermos realizar pesquisas e prosseguir com o desenvolvimento simultaneamente em todos as 3 camadas, pois se torna viável a substituição de um serviço sem interferir nos outros.

As 3 camadas conceituais de conjuntos de serviço que uma internet TCP/IP dispõe são organizadas de forma que sugere dependência entre elas (COMER, Douglas, Interligação de redes com TCP/IP, 2006). De acordo com a figura 2.1 podemos concluir que alicerce de todas e a base é uma camada de serviço de entrega sem conexão. No meio existe uma camada de serviço de transporte confiável da qual a camada de serviços de aplicação depende.

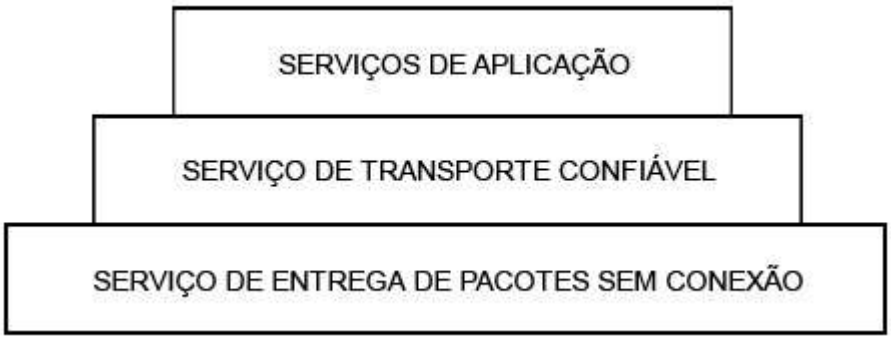

Figura 2.1 - Três camadas conceituais dos serviços Internet.[comer, 2008]

2.2.2. Sistema de entrega sem conexão

Como mostrado na figura 2.1 o serviço mais fundamental de uma internet é o de entrega de pacotes (COMER, Douglas, Interligação de redes com TCP/IP, 2006). Podemos definir esse serviço de um modo técnico como um sistema de entrega de pacote não-confiável, de melhor esforço e sem conexão. O não-confiável vem do fato de que não se pode garantir a sua entrega, o pacote pode sofrer diversas interferências como, por exemplo, ser perdido, duplicado, adiado ou até entregue fora de ordem. Essas condições não irão ser detectadas pelo serviço muito menos informadas ao receptor ou emissor (COMER, Douglas, Interligação de redes com TCP/IP, 2006). Definimos sem conexão, pois cada pacote recebe tratamento independente um dos outros. Quando enviados eles podem seguir diferentes caminhos, serem perdidos ou até mesmo não entregues. E por último, consideramos o serviço como entrega pelo melhor esforço pois “o software de internet faz a melhor tentativa de entregar os pacotes” (COMER, Douglas, Interligação de redes com TCP/IP, 2006).

2.2.3. Finalidade do Internet Protocol

A ferramenta de entrega não-confiável, sem conexão é chamada de Internet Protocol. Como o Internet Protocol encontra-se atualmente na sua versão 4 é comum o chamarmos de IPv4, ou simplesmente IP quando não se tem dúvida

da sua versão. O IP é uma parte tão importante do projeto que a internet em si costuma ser chamada de tecnologia baseada em IP (COMER, Douglas, Interligação de redes com TCP/IP, 2006).

O IP fornece 3 definições de grande importância, a primeira delas é que o IP define a unidade básica de transferência de dados usada numa rede TCP/IP. Dessa maneira, o formato exato de todos os dados é especificado ao passarem pela internet. O IP também define o caminho que um pacote de dados deverá trafegar, dessa maneira fazendo a função de encaminhamento e finalmente o IP define como os pacotes deverão ser processados pelos hardwares de rede e como e quando as mensagens de erro devem ser geradas e os pacotes descartados, dessa maneira provendo um conjunto de regras que incorporam a idéia de entrega não-confiável (COMER, Douglas, Interligação de redes com TCP/IP, 2006).

Esta monografia trata apenas do formato de pacote que o IP específica, ficando a cargo do leitor se aprofundar nos assuntos de encaminhamento de pacotes e tratamento de erros, pois os mesmos não remetem ao escopo do trabalho.

2.2.4. O Datagrama Ipv4

Uma das finalidades do IP é definir a unidade básica de transferência de dados usada numa rede TCP/IP (COMER, Douglas, Interligação de redes com TCP/IP, 2006). Essa unidade básica é chamada de Datagrama Internet, ou simplesmente datagrama IP. Um datagrama é divido em duas partes principais, sendo uma delas o cabeçalho que contém os endereços de origem e destino e um campo para identificar o conteúdo do datagrama. De maneira simplista, o formato geral de um datagrama é como mostrado na figura 2.2 abaixo.

Figura 2.2 - Formato geral de um datagrama IP

O datagrama IP analisado de uma maneira mais detalhada tem o seguinte conteúdo retratado pela figura 2.3, a seguir:

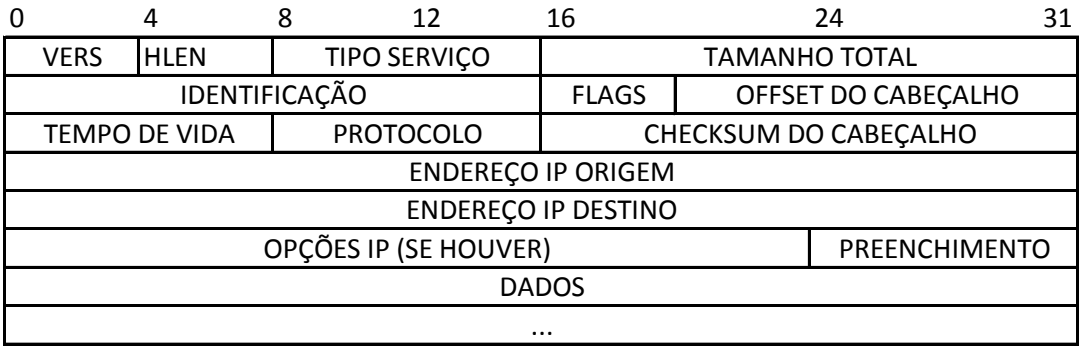

Figura 2.3 - Formato de um datagrama Internet

Como o formato do datagrama IP não remete ao escopo desse trabalho não iremos abranger com maior número de detalhes do mesmo nesta obra. Para mais detalhes e informações sobre a utilização dos campos do datagrama IP podem ser consultadas as obras “COMER, Douglas – Interligação de redes com TCP/IP Vol. 1” e “TANENBAUM, Andrew, *Computer Networks*, 2002”.

2.3. Transmission Control Protocol – TCP

Falamos anteriormente do serviço de entrega de pacotes sem conexão não-confiável e a sua definição no protocolo IP. Neste capitulo, também, iremos tratar do segundo serviço em nível de rede que trata da entrega de fluxo confiável; é o Transmission Control Protocol (TCP) que o define.

2.3.1. A Necessidade da entrega de fluxo

Quando lidamos com o nível mais baixo das redes de comunicação lidamos com a entrega de pacotes não-confiável onde pode ocorrer a perda ou destruição de pacotes devido aos mais variados tipos de erros da rede como, por exemplo, a falha de um hardware ou a sobrecarga da rede. Os sistemas de comutação ainda podem alterar rotas dinamicamente, entregar pacotes fora de

ordem, entregá-los com atraso e ate entregar cópias duplicadas e ainda para alcançar taxas de transferência satisfatórias e eficientes as tecnologias de rede básicas podem definir o tamanho dos pacotes e impor outras restrições.

Quando lidamos com o nível mais baixo, estamos falando das aplicações que se utilizam da rede para enviar um grande volume de dados de um computador pra outro. Para um grande volume de dados sendo transferido não podemos nos dar o luxo de contar com um sistema de remessa não-confiável sem conexão, pois isso se torna incomodo para o lado dos desenvolvedores de sistemas pelo fato de que eles teriam que implementar em cada aplicativo um controle de detecção e correção de erros. Dessa maneira, um dos objetivos quando se pesquisa protocolos de rede tem sido o de encontrar soluções para os problemas de entrega de fluxo confiável (COMER, Douglas, Interligação de redes com TCP/IP, 2006). Assim permitindo que os programadores apenas instanciem o software de protocolo de fluxo que possa ser usado por todos os aplicativos. Desta maneira, um protocolo único de uso geral garante que os aplicativos fiquem isolados dos detalhes de rede e possam ter uma interface uniforme para o serviço de transferência de fluxo.

2.3.2. Fornecendo Confiabilidade

"Como o protocolo pode oferecer transferência confiável se o sistema de comunicação subjacente oferece apenas a remessa de pacotes não-confiável?" (COMER, Douglas, Interligação de redes com TCP/IP, 2006). Esta é a primeira pergunta que vem em mente sempre que garantimos a entrega de um fluxo de dados sem duplicação ou perda de dados pelo serviço de entrega de fluxo confiável.

A resposta pode ser longa e complicada, mas de maneira geral a maioria dos protocolos confiáveis hoje trabalha com uma técnica chamada de confirmação positiva com retransmissão (*Positive Acknowledgement with Retransmission* – PAR) (COMER, Douglas, Interligação de redes com TCP/IP, 2006). Essa técnica consiste em sempre que o destinatário receba dados da origem ele envie de volta uma mensagem de confirmação (ACK). Desta maneira o emissor mantém um registro de cada pacote enviado e aguarda a confirmação antes de enviar o próximo pacote. Para garantir que esse aguardo de mensagem não seja exageradamente demorado o emissor também gera um timer para cada pacote. Assim se o timer expirar e a confirmação não chegar do destinatário, o emissor re-envia esse pacote (COMER, Douglas, Interligação de redes com TCP/IP, 2006). A figura 2.4 mostra um exemplo de um protocolo utilizando PAR, o emissor envia os pacotes representados pelas linhas e a distância vertical representa o tempo entre

cada transmissão. Podemos ver claramente que o emissor só envia o pacote 2 depois da confirmação de recebimento do pacote 1 pelo receptor.

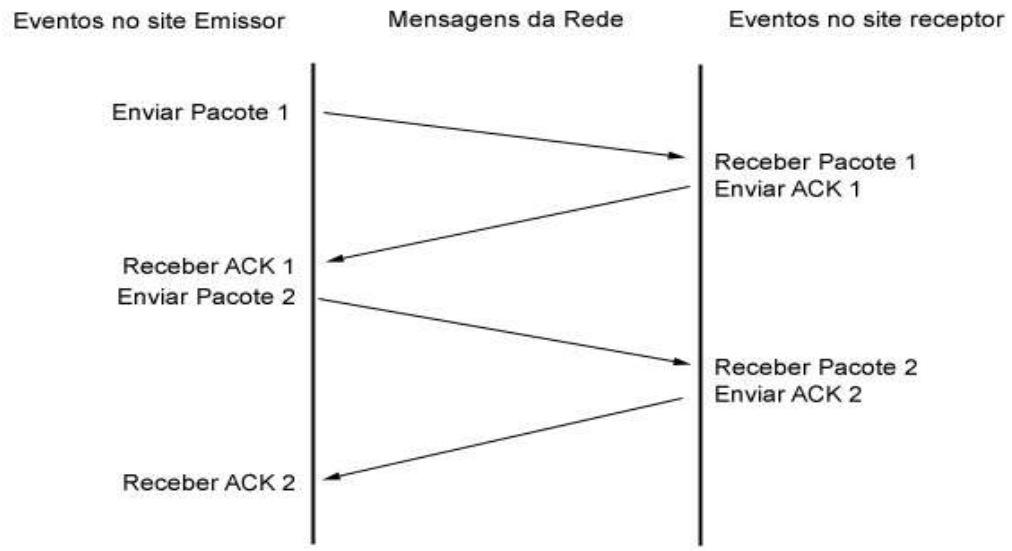

Figura 2.4 - Protocolo Utilizando PAR,[Comer, 2008]

Já a figura 2.5 exemplifica a expiração do timer do pacote, o pacote 1 que deveria chegar ao receptor não chega por algum motivo, logo o receptor não envia um ACK sinalizando o recebimento do pacote. as linhas pontilhadas representam o tempo que seria gasto pela transmissão e a confirmação de um pacote se o mesmo não tivesse sido perdido.. Quando o timer expira e o emissor não recebeu o ACK ele re-envia o pacote 1.

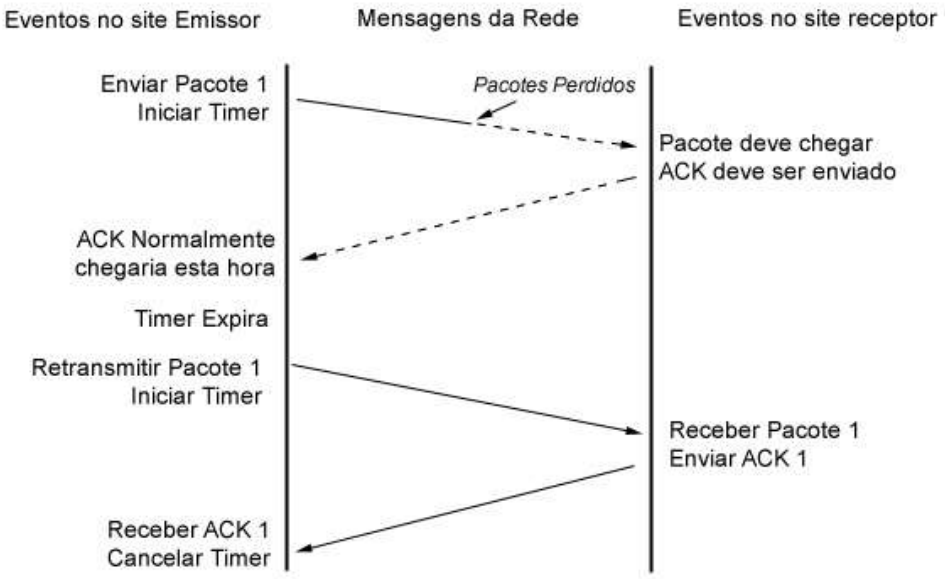

Figura 2.5 - Exemplo de timeout e retransmissão [Comer, 2008]

Os mecanismos citados anteriormente garantem que o pacote não seja perdido, mas o que aconteceria no caso de um pacote duplicado? Suponha que a rede esteja sofrendo um alto trafego com isso se tornando sobrecarregada e gerando um alto delay na entrega dos pacotes. O timer iria expirar e o emissor iria enviar um segundo pacote prematuramente gerando um pacote duplicado. Para resolver esse problema os protocolos confiáveis atribuem números de seqüência a cada pacote e exigem que o receptor se lembre quais números de seqüência ele recebeu. Da mesma maneira a confirmação de entrega do receptor deve conter o número do pacote recebido para poder ser efetuado o controle por parte do emissor de confirmações por pacote.

2.3.3. Janelas Deslizantes

Se pensarmos com cuidado nos mecanismos anteriores iremos chegar a conclusão de que eles são eficientes quando se trata de manter o fluxo confiável, mas vemos que eles tem uma falha no quesito de utilização da largura de banda da rede. Podemos nomear o método anterior como de confirmação como um protocolo de confirmação positiva simples. Nele temos vários problemas quando se trata de aproveitamento da largura de banda, primeiro, olhando a figura 2.5 podemos observar que só existe comunicação em uma direção. O emissor transmite o pacote e aguarda por uma confirmação do destinatário antes de transmitir outros; isso acontece mesmo que a rede tenha capacidade para comunicação simultânea. Durante os períodos de tempo em que as máquinas atrasam as respostas a rede permanece completamente ociosa. Para enxergarmos o problema com maior clareza consideremos uma rede onde exista um tráfego intenso, nela poderíamos ter atrasos de transmissão maiores ainda. (TANENBAUM, Andrew, *Computer Networks*, 2002).

Para contornar o problema de desperdício de largura de banda dos protocolos de confirmação positiva simples o TCP emprega um conceito adicional que está por trás da transmissão de fluxo, a técnica de janela deslizante (TANENBAUM, Andrew, *Computer Networks*, 2002).

Os protocolos que utilizam o método de confirmação de janela deslizante conseguem um aproveitamento bem melhor da largura de banda, isso acontece devido ao fato de que o emissor pode emitir vários pacotes antes de esperar uma confirmação.

Para entender o funcionamento da operação de janela deslizante imaginamos uma seqüência de pacotes que devem ser enviados de acordo com a figura 2.6. O protocolo utiliza uma janela pequena de tamanho fixo para delimitar os pacotes; então todos os pacotes que estão dentro da janela são enviados. De acordo com a nossa figura o emissor irá emitir 8 pacotes antes de receber uma confirmação, e então assim que o receptor receber a confirmação para o primeiro pacote dentro da janela, ela é deslizada e então é transmitido o próximo pacote. A janela continua deslizando de acordo com as confirmações recebidas. Na figura 2.7 temos um exemplo de três pacotes sendo transmitidos usando um sistema de janela deslizante, podemos visualizar que o emissor envia o próximo pacote na janela antes de receber a confirmação do anterior.

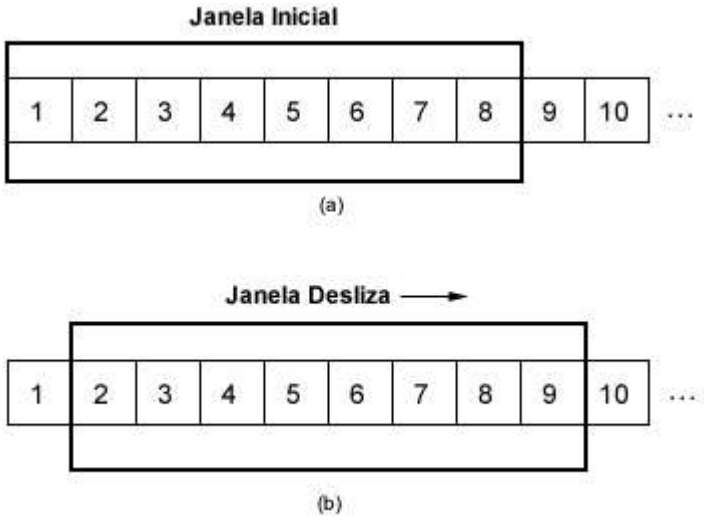

Figura 2.6 - Exemplo de janela deslizante.[Comer, 2008]

Logo o desempenho dos protocolos de janela deslizante está atrelado ao tamanho da janela que irá ser utilizada e à velocidade que a rede aceita pacotes (COMER, Douglas, Interligação de redes com TCP/IP, 2006). Assim, se aumentarmos o tamanho da janela para um valor bem ajustado podemos saturar a rede por completo e acabar com o tempo ocioso da mesma, fazendo com que o emissor possa transmitir pacotes tão rapidamente quanto a rede possa transferi-los.

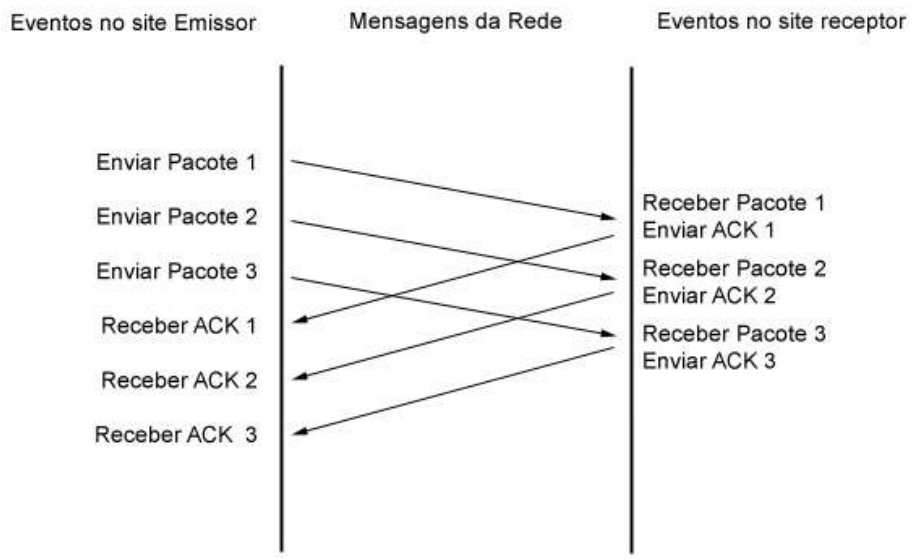

Figura 2.7 - Exemplo de protocolo de janela deslizante.[Comer, 2008]

O TCP utiliza um mecanismo de janela deslizante especializado, que difere do protocolo de janela deslizante simplificado citado, pelo fato do TCP utilizar janelas de tamanho variável. Esse mecanismo especializado do TCP não remete ao escopo desta obra. Para mais detalhes consultar COMER, Douglas, Interligação de redes com TCP/IP, 2006 ou TANENBAUM, Andrew, *Computer Networks*, 2002.

2.3.4. Formato do Segmento TCP

O TCP enxerga o fluxo de dados como uma seqüência de octetos ou bytes que ele divide em segmentos para a transmissão (COMER, Douglas, Interligação de redes com TCP/IP, 2006). Estes segmentos são a unidade de

transferência entre duas máquinas que utilizam o TCP e é utilizado para se estabelecer conexões, transferir dados, enviar confirmações, anunciar tamanhos de janela e fechar conexões.

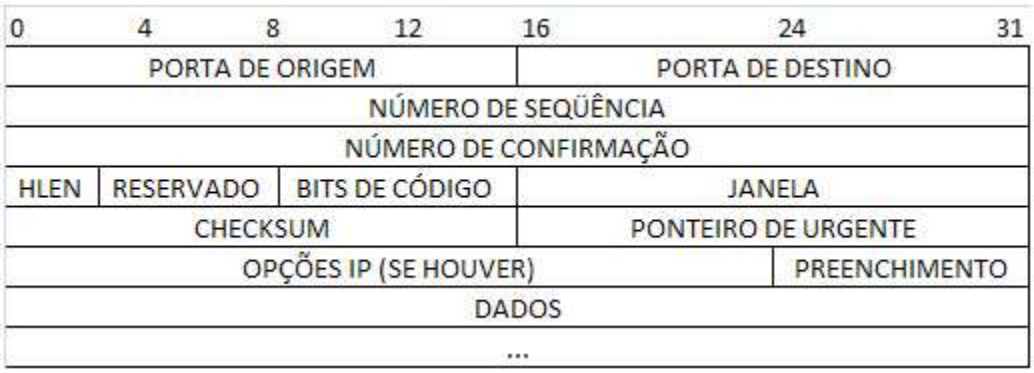

Figura 2.8 - Formato de um segmento TCP [Tanembaum, 2008]

Como mostra a figura 2.8 “Cada segmento pode ser dividido em duas partes: um cabeçalho seguido por dados” (COMER, Douglas, Interligação de redes com TCP/IP, 2006). Para o escopo desta obra o campo de maior importância é o campo de número de seqüência (ISN), este campo foi escolhido devido ao fato de ser um campo de 32 bits, desta maneira podemos armazenar de forma esteganográfica um número grande sem nos preocuparmos com restrições de tamanho e este campo conta com a vantagem de receber valores randômicos, logo não tendo nenhum valor padrão, tornando assim a identificação de sua alteração difícil. O campo do número de seqüência identifica a posição no fluxo de bytes do emissor de dados no segmento. Isso acontece pois o TCP envia dados em segmentos de tamanho variável, logo os segmentos retransmitidos podem

incluir mais dados do que o original, desta maneira o TCP não pode ter suas confirmações referidas apenas a datagramas ou segmentos e sim a uma posição no fluxo de dados usando um número de seqüência. Os números de seqüência são usados para reconstruir o fluxo de dados do lado do receptor. Como dito anteriormente, os segmentos podem ser perdidos e entregues fora de ordem. Logo, com o número de seqüência o receptor pode re-construir o fluxo desde o início. Essa confirmação é feita especificando um valor a mais de seqüência na posição de octeto mais alta no prefixo contíguo que recebeu. Desta maneira pode-se concluir que “uma confirmação TCP especifica o número de seqüência do próximo octeto que o receptor espera receber” (TANENBAUM, Andrew, *Computer Networks*, 2002).

Para este projeto iremos utilizar os serviços da rede TCP/IP, sem conexão, que utilizam a pilha TCP/IP e utilizaremos especificamente o campo ISN do TCP para realizar a esteganografia da mensagem criptografada. A criptografia será utilizada pois os dados transmitidos no cabeçalho TCP não são seguros. De maneira que qualquer pessoa que capturar o pacote na rede poderá ter acesso as informações contidas neste cabeçalho em texto inteligível. O capítulo a seguir apresenta as técnicas de criptografia e esteganografia.

3. CAPÍTULO 3 - ESTEGANOGRAFIA E CRIPTOGRAFIA

Desde antes da idade moderna já existiam técnicas de esteganografia e criptografia, com o advento da tecnologia e da computação essas técnicas passaram por modificações severas até se tornarem os mecanismos de segurança e proteção que são utilizados hoje por computadores em todo o mundo.

Neste capítulo são discutidos os conceitos de esteganografia moderna, criptografia abordando as técnicas simétricas e assimétricas e com maior profundidade o algoritmo de criptografia assimétrica RSA.

3.1. Esteganografia

A esteganografia é uma técnica que tem por objetivo permitir escrever mensagens escondidas de uma maneira que ninguém, além do remetente e do destinatário, suspeite da existência desta mensagem (WAYNER, Peter, Disappearing cryptography: information hiding: steganography & watermarking, 2002).

A distinção entre esteganografia e criptografia se dá pelo fato de que a criptografia tem por objetivo esconder o conteúdo da mensagem, mas não

esconder a existência de tal mensagem (STALLINGS, William, Criptografia e Segurança de Redes, 2008).

A existência das técnicas esteganográficas datam de antes de Cristo, mas hoje em dia o termo inclui as técnicas de ocultamento de informação digital na camada de transporte, podendo ser desde arquivos ou até mesmo de um protocolo como é tratado neste trabalho (WAYNER, Peter, Disappearing cryptography: information hiding: steganography & watermarking, 2002).

Muitas técnicas de esteganografia foram usadas historicamente, dentre as mais famosas podemos citar a marcação de caractere que era feita com pequenos furos em cima das letras selecionadas e estas só podiam ser vistas quando o papel se encontrava sobre um fonte de luz e diversos tipos de tintas invisíveis que só apareciam quando algum reagente era aplicado no papel (STALLINGS, William, Criptografia e Segurança de Redes, 2008).

Atualmente as técnicas mais conhecidas de esteganografia são aquelas em que é alterado o bit menos significativo de cada pixel de uma imagem como é proposto em (WAYNER, Peter, Disappearing cryptography: information hiding:

steganography & watermarking, 2002). Desta maneira é possível esconder o equivalente a 2,3 megabytes de informação numa imagem de 2048 x 3072 pixels.

Neste trabalho é proposto um método de se esconder uma mensagem no cabeçalho do protocolo TCP, no campo de seu número seqüencial, fazendo com que na verdade o que seria apenas um mecanismo de controle do protocolo passe a transmitir uma mensagem para um recipiente.

Apesar da esteganografia esconder a mensagem enviada, uma vez que o método esteganográfico é descoberto ele se torna inútil (STALLINGS, William, Criptografia e Segurança de Redes, 2008). Uma maneira de contornar esse problema é codificar uma mensagem de maneira que se o método esteganográfico for quebrado, o conteúdo da mensagem não seja descoberto, assim podemos utilizar um método de criptografia aliado a esteganografia

3.2. Criptografia

A criptografia consiste no “uso de um algoritmo matemático para transformar os dados em um formato que não seja prontamente decifrável. A transformação e, subseqüente, recuperação dos dados depende de um algoritmo e de zero ou mais chaves de criptografia” (STALLINGS, 2008, p.11).

A criptografia moderna pode ser divida em dois grandes ramos, a criptografia simétrica ou de chave privada e a criptografia de chave pública ou assimétrica (KATZ, Jonathan e LINDELL, Yehuda, Introduction to Modern Cryptography, 2007),

3.2.1. Criptografia Simétrica

O modelo de criptografia simétrica, também chamada de criptografia convencional, utiliza um modelo de criptossistema em que a operação de criptografia e decriptografia utilizam a mesma chave (STALLINGS, William, Criptografia e Segurança de Redes, 2008).

Um esquema do modelo de cifra simétrica pode ser visualizado na figura 3.1

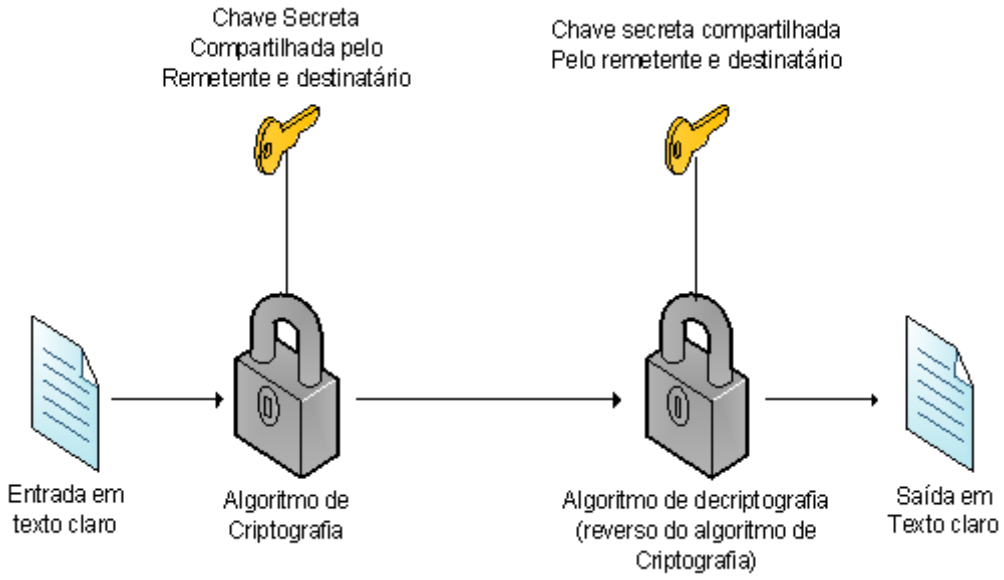

Figura 3.1 – Modelo simplificado da criptografia convencional [Stallings, 2008]

A figura 3.1 mostra um esquema de cifra simétrica que possui cinco elementos:

- **Texto Claro**: É a mensagem ou dados em seu formato original, sem estarem cifrados e de forma inteligível, são utilizados como entrada para o algoritmo de criptografia.
- **Algoritmo de Criptografia**: O algoritmo matemático responsável por fazer substituições e transformações de maneira a tornar o texto claro em texto cifrado.

- **Chave Secreta**: A chave secreta também é utilizada como entrada para o algoritmo de criptografia. A chave secreta independe dos conteúdos do texto claro e do algoritmo. O algoritmo realizará suas substituições e transformações baseado na chave secreta fornecida como entrada.
- **Texto Cifrado**: O texto cifrado consiste da saída de dados gerada pelo algoritmo de criptografia e da chave secreta. Se forem utilizadas chaves diferentes para a mesma mensagem, os textos cifrados gerados também serão diferentes entre si. Neste formato, o texto encontra-se ininteligível.
- **Algoritmo de decriptografia:** Algoritmo responsável por produzir o texto claro original utilizando a chave correspondente.

Para se utilizar a criptografia convencional de maneira segura é necessário se ter dois requisitos: Um algoritmo de criptografia forte e manter a chave secreta de forma segura entre o receptor e o emissor.

Um algoritmo de criptografia forte garante que um indivíduo malicioso que conheça o algoritmo e tenha acesso a textos cifrados gerados por esse algoritmo não consiga decifrar o texto cifrado ou descobrir a chave secreta utilizada. Pode-se ainda enfatizar este requisito de uma maneira mais robusta, de forma a garantir que o indivíduo malicioso não consiga decriptografar o texto cifrado ou descobrir a

chave secreta, mesmo tendo acesso a diversos textos cifrados gerados pelo algoritmo, juntamente com os seus respectivos pares de texto claro.

Os elementos essenciais de um esquema de criptografia simétrico podem ser visualizados na figura 3.2.

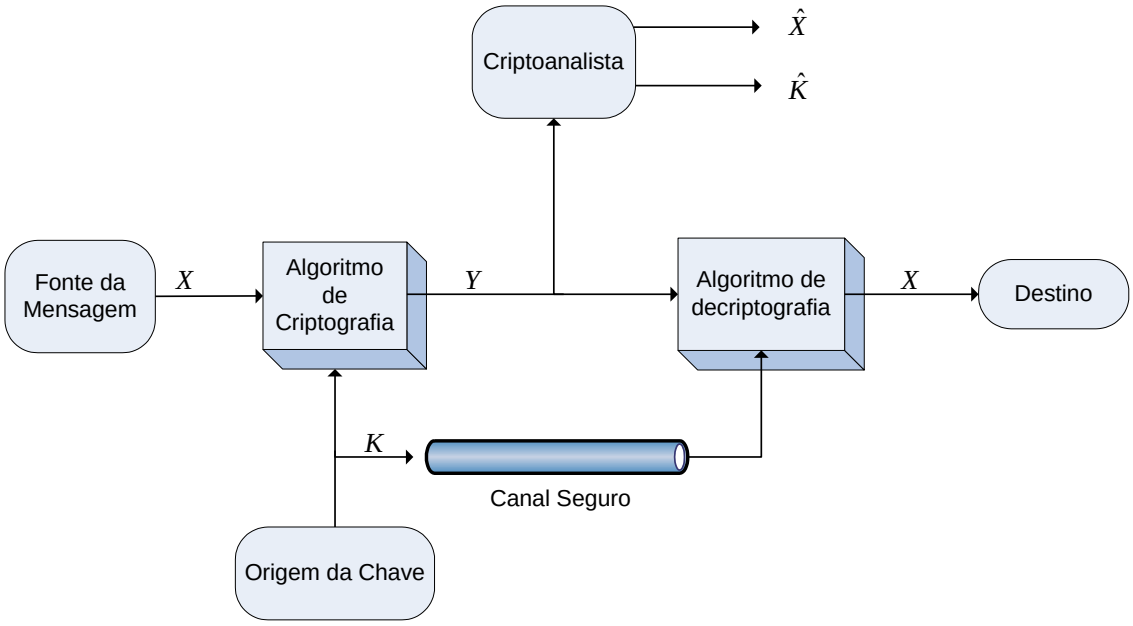

Figura 3.2 – Modelo de criptossistema convencional [Stallings, 2008]

A figura 3.2 denota uma mensagem de texto claro, $X = X_1, X_2, ..., X_M$, produzida na origem. Considera-se que os M elementos de X sejam letras do alfabeto finito, composto por 26 letras maiúsculas. Vale ressaltar que para os

arquivos digitais é utilizado o alfabeto binário $\{0,1\}$. Ainda, na origem é gerada uma chave na forma $K = K_1, K_2, ..., K_J$. Como essa chave foi gerada na origem, ela precisa ser transmitida para o destino por um canal seguro.

De posse da chave K e o texto X gerados na entrada, o algoritmo de criptografia forma um texto cifrado $Y = Y_1, Y_2, ..., Y_N$.

Desta maneira:

$$Y = E(K, X) \quad \textbf{(3.1)}$$

Que significa que Y é produzido utilizando-se um algoritmo de criptografia E como função de um texto claro X, com a função específica determinada pelo valor da chave K.

Logo, o processo pode ser invertido e o texto decifrado pelo receptor de posse da chave K utilizando o algoritmo de decriptografia D:

$$X = D(K, Y) \quad \textbf{(3.2)}$$

A partir dessas ponderações pode-se concluir que a criptografia simétrica tem um problema de distribuição de chaves, pois a criptografia simétrica requer que dois computadores já compartilhem uma chave que de alguma forma foi

distribuída a eles ou que se utilize um centro de distribuição de chaves. (KATZ, Jonathan e LINDELL, Yehuda, Introduction to Modern Cryptography, 2007).

Com a idéia de resolver esse problema, começou a se estudar um novo tipo de criptografia que iria revolucionar a história da criptografia, este tipo de criptografia é conhecido hoje como criptografia de chave pública.

3.2.2. Criptografia de Chave-Pública

Como dito no item anterior, a criptografia de chave pública foi a grande revolução na história da criptografia, primeiro porque ela muda o paradigma dos algoritmos criptográficos que antes faziam apenas substituição e permutas para algoritmos baseados em funções matemáticas, além disso o mais importante é que a criptografia de chave-pública é assimétrica, sendo assim ela envolve o uso de duas chaves separadas, diferente da criptografia simétrica que utiliza apenas uma chave. A utilização de duas chaves provoca grandes impactos nas áreas de confidencialidade, distribuição de chaves e autenticação conforme veremos neste capítulo (STALLINGS, William, Criptografia e Segurança de Redes, 2008).

Os algoritmos assimétricos possuem uma característica de suma importância:

- Detendo apenas o conhecimento do algoritmo de criptografia e da chave de criptografia é computacionalmente inviável determinar a chave de decriptografia.

O algoritmo assimétrico RSA, tratado neste trabalho, possui ainda uma segunda característica:

- Pode-se usar qualquer uma das chaves relacionadas para a criptografia, com a outra sendo usada para decriptografia.

A criptografia de chave-pública pode ser representada por um esquema de seis elementos como mostra a figura 3.3:

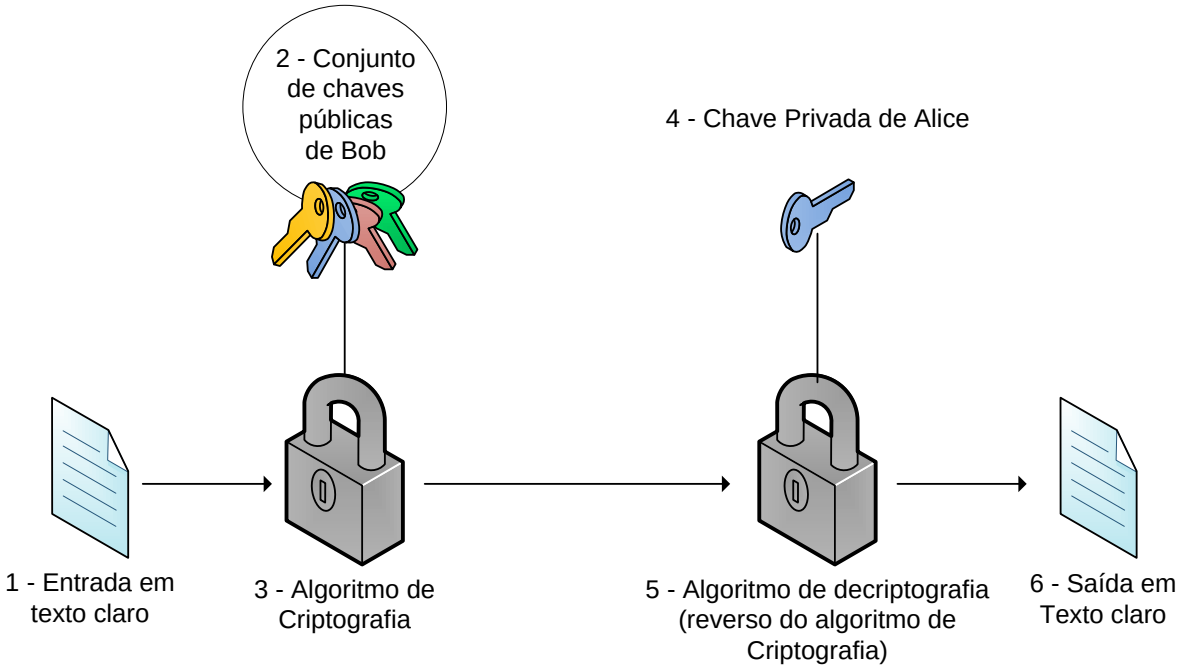

Figura 3.3 - Esquema de criptografia de chave pública [Stallings, 2008]

- **Texto Claro:** É a mensagem ou dados em seu formato original, sem estarem cifrados e de forma inteligível, são utilizados como entrada para o algoritmo de criptografia
- **Algoritmo de Criptografia:** O algoritmo responsável por fazer as alterações no texto claro.
- **Chaves Pública e Privada:** Um par de chaves selecionado de modo que, se uma for usada para criptografia, a outra será usada para decriptografia. As transformações feitas no texto iram depender da chave que será fornecida como entrada.
- **Texto cifrado:** O texto cifrado consiste da saída de dados gerada pelo algoritmo de criptografia e da chave secreta. Se forem utilizadas chaves diferentes para a mesma mensagem, o textos cifrados gerados também serão diferentes entre si. Neste formato o texto encontra-se ininteligível.
- **Algoritmo de decriptografia:** Algoritmo responsável por produzir o texto claro original utilizando a chave correspondente.

Dentro deste esquema possuímos 4 etapas essenciais:

1. É gerado um par de chaves por cada usuário que deverá ser usado para criptografar e decriptografar as mensagens.

2. Cada usuário distribui sua chave pública de maneira que achar mais apropriada, esta maneira pode ser desde um registro público ou outro arquivo acessível. A outra chave é mantida privada com o usuário. Cada usuário mantém o seu conjunto de chaves públicas obtidas de outros usuários.
3. Caso Bob queira enviar uma mensagem confidencial para Alice, este deve usar a chave pública de Alice para criptografar a mensagem.
4. Quando a mensagem é recebida por Alice, esta deve utilizar sua chave privada para decriptografar a mensagem. Desta maneira nenhum outro usuário pode decifrar a mensagem, pois apenas Alice conhece sua chave privada.

Dessa maneira, as chaves privadas nunca precisam ser distribuídas, pois são geradas localmente pelos participantes, e todos os participantes têm acesso às chaves públicas. A comunicação será protegida enquanto a chave privada do usuário for mantida em sigilo.

Podemos fazer uma análise mais aprofundada dos elementos essenciais do esquema de criptografia de chave pública utilizando a figura 3.4.

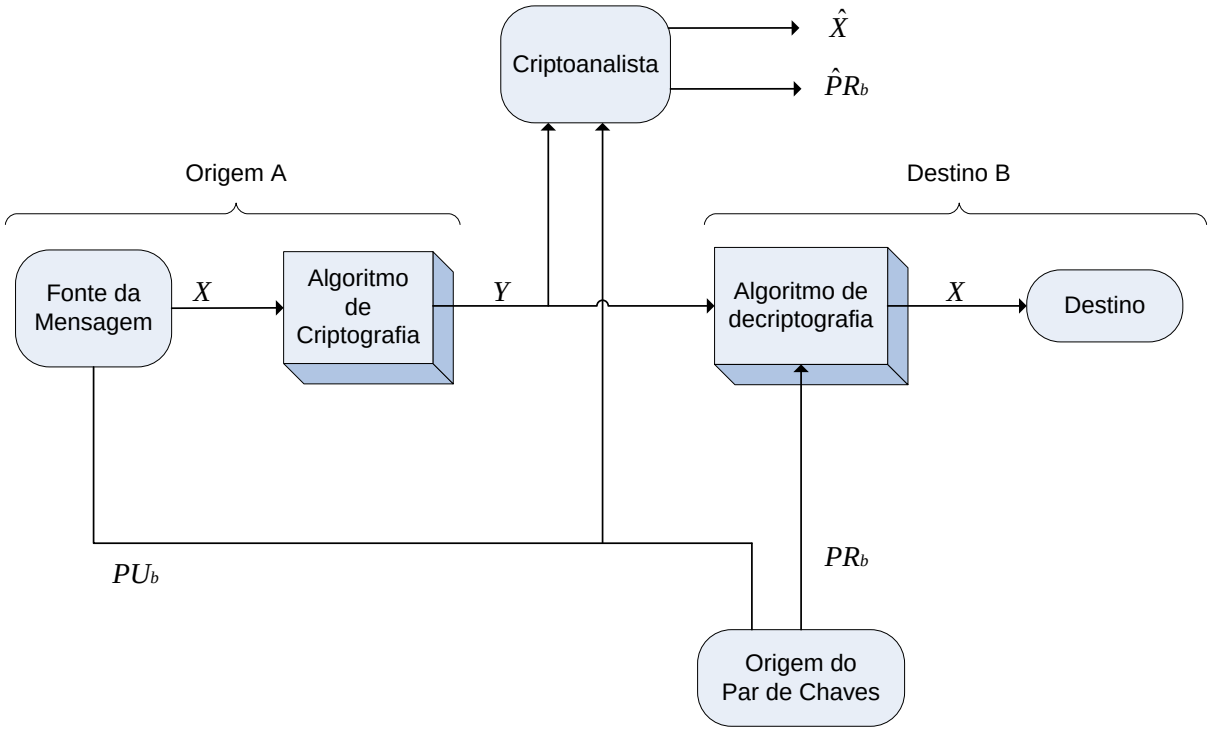

Figura 3.4 - Modelo de Criptossistema de Chave Pública [Stallings, 2008]

A figura 3.4 mostra uma mensagem em texto claro, $X = X_1, X_2, ..., X_M$, sendo que os M elementos de X são letras em algum alfabeto finito. Esta mensagem tem como destino B. B por sua parte gera um par de chaves, sendo uma chave pública, PU_b, e uma chave privada, PR_b. A chave privada PR_b é conhecida apenas por B, enquanto a chave pública PU_b é distribuída publicamente, dessa maneira ficando acessível por A.

Utilizando a mensagem X e a chave pública de criptografia PU_b como entrada, A gera o texto cifrado $Y = Y_1, Y_2, ..., Y_N$:

$$Y = \mathrm{E}(PU_b, X) \quad \textbf{(3.4)}$$

Utilizando a chave privada correspondente, o receptor da mensagem é capaz de inverter a transformação:

$$Y = \mathrm{D}(PR_b, Y) \quad \textbf{(3.5)}$$

O criptossistema ilustrado na figura 3.4 depende de um algoritmo criptográfico baseado em duas chaves públicas relacionadas, apesar de hoje existirem esses algoritmos, em 1976 eles ainda não eram realidade. Diffie e Hellman fizeram os postulados desses algoritmos e estabeleceram as condições a que esses algoritmos precisam atender em DIFFIE, W. e HELLMAN, M. "*Multiuser cryptographic techniques*". *IEEE Transactions on Information Theory*. 1976, elas são as seguintes:

1. Ser computacionalmente fácil, permitindo assim que uma parte B gere um par de chaves (chave pública PU_b, chave privada PR_b).
2. Ser computacionalmente fácil, permitindo assim que um emissor A, de posse da chave pública e da mensagem a ser criptografada, M, gere o texto cifrado correspondente:

$$C = \mathrm{E}(PU_b, M) \quad \textbf{(3.6)}$$

3. Ser computacionalmente fácil, permitindo assim que um receptor B decriptografe o texto cifrado, C, utilizando a chave privada para recuperar a mensagem original.

$$M = \mathrm{D}(PR_b, C) = \mathrm{D}(PR_b, \mathrm{E}(PU_b, M)) \quad \textbf{(3.7)}$$

4. Não ser computacionalmente viável, para um usuário malicioso que conhece a chave pública, PU_b, determinar a chave privada PR_b.

5. Não ser computacionalmente viável, para um usuário malicioso que conhece a chave pública, PU_b, e um texto cifrado, C, recuperar a mensagem original M.

6. Poder aplicar as duas chaves em qualquer ordem:

$$M = \mathrm{D}(PU_b, \mathrm{E}(PR_b, M)) = \mathrm{D}(PR_b, \mathrm{E}(PU_b, M)) \quad \textbf{(3.8)}$$

Estes requisitos são basicamente uma função unidirecional com segredo. Define-se uma função unidirecional como aquela que mapeia um domínio em um intervalo de modo que todo valor da função tem um inverso único, sendo que o cálculo da função seja fácil, mas o cálculo do inverso seja inviável.

$Y = \mathrm{f}(X)$ Fácil

$X = \mathrm{f}^{-1}(Y)$ Inviável

Considera-se fácil um problema que pode ser resolvido em tempo polinomial como uma função do tamanho da entrada, ou seja, se temos uma entrada de tamanho n bits, o tempo para calcular a função será proporcional a n^a, onde a é uma constante fixa. Para definir um problema como inviável, se diz geralmente que o esforço para solucionar o problema deve aumentar de maneira mais rápida do que o tempo polinomial como uma função do tamanho de entrada.

Sendo assim, se tivermos uma entrada de tamanho n bits e o tempo para calcular a função for proporcional a 2^n, o problema seria considerado inviável.

Com o conhecimento acima pode-se definir o que seria uma função unidirecional com segredo, que é fácil de calcular numa direção e inviável na outra,a não ser que certa informação adicional seja conhecida. Com esta informação pode-se calcular o inverso em tempo polinomial.

Logo uma função polinomial com segredo pode ser definida como uma família de funções reversíveis f_k, tal que:

$Y = \mathrm{f}_k(X)$ fácil, se k e X forem conhecidos

$X = \mathrm{f}_k^{-1}(Y)$ fácil se k e Y forem conhecidos

$X = \mathrm{f}_k^{-1}(Y)$ inviável se Y for conhecido, mas k não for conhecido

Portanto, para se desenvolver um esquema de chave pública prática é necessária a descoberta de uma função unidirecional com segredo adequada.

Desta maneira podemos examinar mais de perto os elementos de um esquema de criptografia de chave pública, para termos uma idéia do funcionamento real de um algoritmo de chave pública, na próxima seção será

estudado o algoritmo RSA, que é utilizado como tema deste trabalho para a criptografia e decriptografia de dados.

3.2.3. O Algoritmo RSA

Tudo que se precisava para desenvolver um algoritmo de chave pública foi levantado no artigo de Diffie e Hellman em 1976, uma das primeiras respostas foi desenvolvida em 1977 no *Massaschusets Institute of Technology*, MIT, por Ron Rivest, Adi Shamir e Len Adleman e foi nomeada como o esquema Rivest-Shamir-Adleman (RSA) e desde então tem sido a principal técnica de criptografia de chave pública.

O algoritmo RSA criptografa os dados em blocos de texto utilizando uma expressão com exponenciais. Para isso são utilizados dois blocos, um bloco de texto claro M e um bloco de texto cifrado C. Cada um desses blocos deve ser menor ou igual a $\log_2(n)$ de maneira a tornar o tamanho do bloco igual a i bits, tal que $2^i < n \leq 2^{i+1}$.

Desta maneira temos:

$$C = M^e \bmod n$$

$$M = C^d \bmod n = (M^e)^d \bmod n = M^{ed} \bmod n$$

Conseguimos então visualizar um algoritmo de chave pública, pois apenas o emissor conhece o valor de e, e somente o receptor conhece o valor de d. Logo podemos definir a chave pública como $PU = \{e, n\}$ e a chave privada como $PR = \{d, n\}$.

De acordo com STALLINGS, William, Criptografia e Segurança de Redes, 2008, para que o algoritmo RSA seja satisfatório é necessário que três requisitos sejam atendidos:

1. Ser possível encontrar valores de e, d, n tais que $M^{ed} \bmod n = M$ para todo $M < n$.
2. Ser relativamente fácil de calcular $M^e \bmod n$ e $C^d \bmod n$ para todos os valores de $M < n$.
3. Ser inviável determinar d sabendo e e n.

Não é do escopo deste trabalho abordar a validação destes requisitos. Iremos considerar que os 3 requisitos são atendidos plenamente. As demonstrações que comprovam que o RSA atende tais requisitos podem ser encontradas no Capítulo 9 e no apêndice 9A em STALLINGS (Stallings, 2008).

Para podemos formular o esquema RSA são necessários alguns fatores. Estes fatores podem ser tanto privados como públicos e escolhidos ou calculados, eles são os seguintes:

1. Dois números primos p, q privados e escolhidos.
2. $n = pq$, público e calculado.
3. e com $mdc(\phi(n), e) = 1; 1 < e < \phi(n)$, público e escolhido
4. $d \equiv e^{-1} (\mathrm{mod} \phi(n))$ privado e calculado.

Iremos utilizar um exemplo do algoritmo RSA que se encontra em Singh, S, *The code book: the science of secrecy from ancient egypt to quantum cryptography*. Nova Iorque, 1999 (Singh, 1999). Iremos fazer o passo a passo da criptografia, decriptografia e geração de chaves. Deve ser ressaltado que para simplificação são utilizados números primos pequenos, mas para questões de seguranças os números primos gerados em algoritmos RSA reais devem ser números grandes.

Primeiro iremos gerar as chaves pública e privada. Para isso fazemos:

1. Seleção de dois números primos p e q, onde $p=17$ e $q=11$, apenas utilizados como exemplo.
2. Fazemos o calculo de $n=pq=17\times11=187$.
3. Com estes valores podemos então calcular $\phi(n)$ que é igual a $(p-1)(q-1)$. Logo $\phi(n)=16\times10=160$.
4. Precisamos de um numero e que seja relativamente primo a $\phi(n)=160$, ou seja, $mdc(\phi(n),e)=1;1<e<\phi(n)$. Escolheremos $e=7$.
5. Utilizando o algoritmo estendido de Euclides e sabendo que $d\equiv e^{-1}(\mathrm{mod}\phi(n))$ podemos então finalmente calcular d de maneira q $de\equiv1(\mathrm{mod}160)$ e $d<160$. Sendo assim o valor correto para d seria de 23, pois $23\times7=161=1\times160+1$.

Com estes valores agora estamos de posse das chaves pública e privada. Temos a chave pública $PU=$ { 7 , 187 } e a chave privada $PR=$ { 23 , 187 }. Agora podemos finalmente criptografar uma entrada de texto claro. Iremos utilizar uma entrada de texto claro $M=88$ e então gerar o texto cifrado $C=M^e \bmod n$. Desta maneira $C=88^7 \bmod 187$. Sabendo as propriedades de aritmética modular podemos efetuar tal cálculo da seguinte maneira:

$$88^7 \bmod 187 = 88^4 \bmod 187 \times 88^2 \bmod 187 \times 88^1 \bmod 187 \bmod 187$$

$$88^1 \bmod 187 = 88$$

$$88^2 \bmod 187 = 7744 \bmod 187 = 77$$

$$88^4 \bmod 187 = 59.969.536 \bmod 187 = 132$$

$$88^7 \bmod 187 = 88 \times 77 \times 132 \bmod 187 = 894.432 \bmod 187 = 11$$

Logo nosso texto criptografado utilizando o algoritmo RSA e a chave pública $PU = \{ 7 , 187 \}$ é o bloco de texto cifrado $C = 11$.

De maneira similar realizamos a decriptografia utilizando a chave privada $PR = \{ 23 , 187 \}$. Já sabemos que $M = C^d \bmod n$, logo $M = 11^{23} \bmod 187$. Mais uma vez utilizando as propriedades de aritmética modular calculamos:

$$11^{23} \bmod 187 = \left[11^1 \bmod 187 \times 11^2 \bmod 187 \times 11^4 \bmod 187 \times 11^8 \bmod 187 \times 11^8 \bmod 187 \right] \bmod 187$$

$$11^1 \bmod 187 = 11$$

$$11^2 \bmod 187 = 121$$

$$11^4 \bmod 187 = 14.641 \bmod 187 = 55$$

$$11^8 \bmod 187 = 214.358.881 \bmod 187 = 33$$

$$11^{23} \bmod 187 = 11 \times 121 \times 55 \times 33 \times 33 \bmod 187 = 79.720.245 \bmod 187 = 88$$

Finalmente obtemos nosso resultado de texto claro $M = 88$ assim realizando a decriptografia utilizando o algoritmo RSA.

4. CAPÍTULO 4 - VISÃO GERAL DO PROJETO

Este capítulo trata da descrição geral do projeto e seu funcionamento conforme a topologia e fluxograma apresentados abaixo.

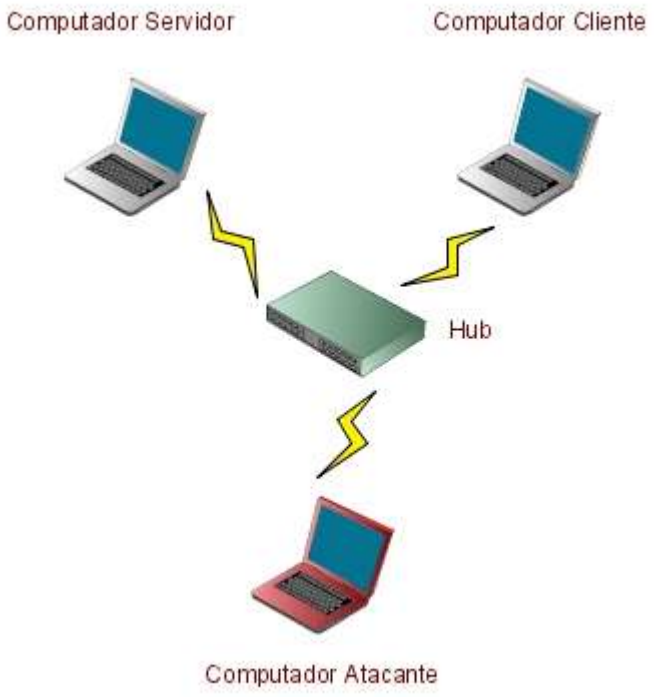

Figura 4.1 - Topologia do Projeto

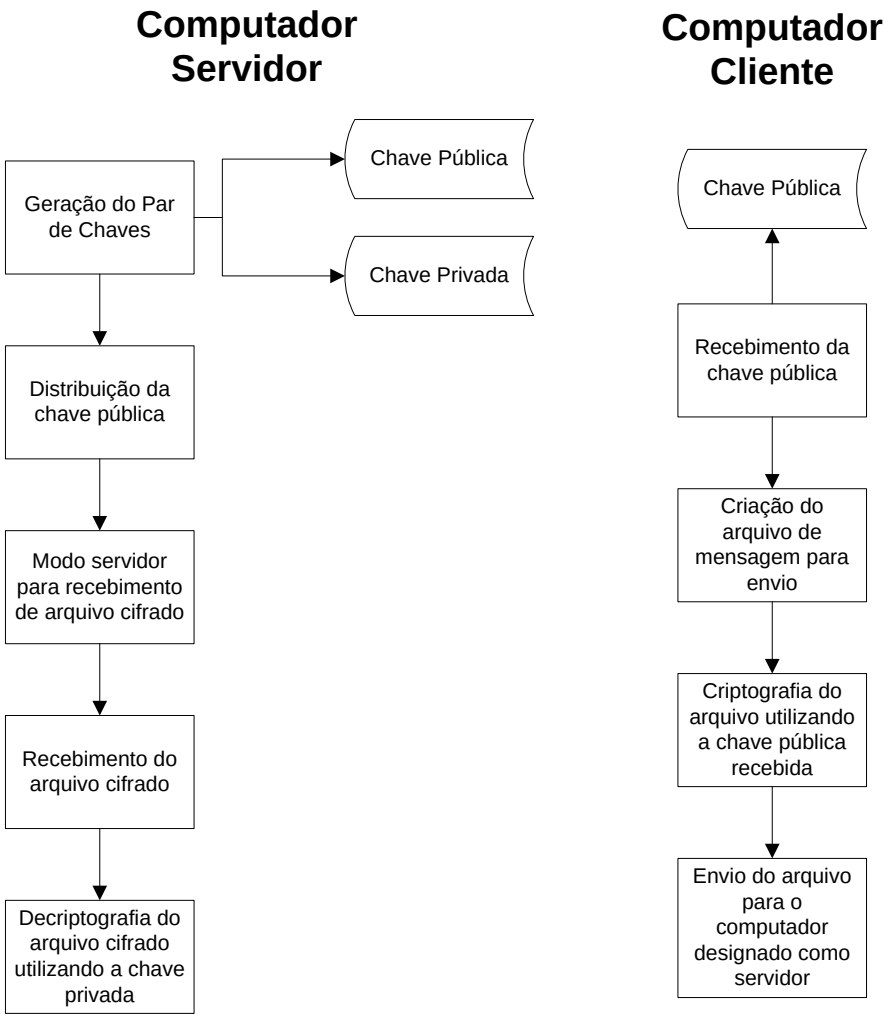

Figura 4.2 - Fluxograma dos processos dos computadores cliente e servidor

4.1. Servidor

No computador representado como servidor na topologia serão realizadas quatro atividades. Primeiro, irão ser geradas um par de chaves RSA, uma pública e uma privada. Num segundo momento o servidor irá distribuir a chave pública de alguma maneira para outros computadores para poder receber as mensagens

devidamente criptografadas. Então, o servidor ficara em modo de espera aguardando o recebimento da mensagem, e finalmente realizará a decriptografia dos dados.

Todos estes passos são realizados por um menu interativo com o usuário como mostrado na figura 4.2 abaixo.

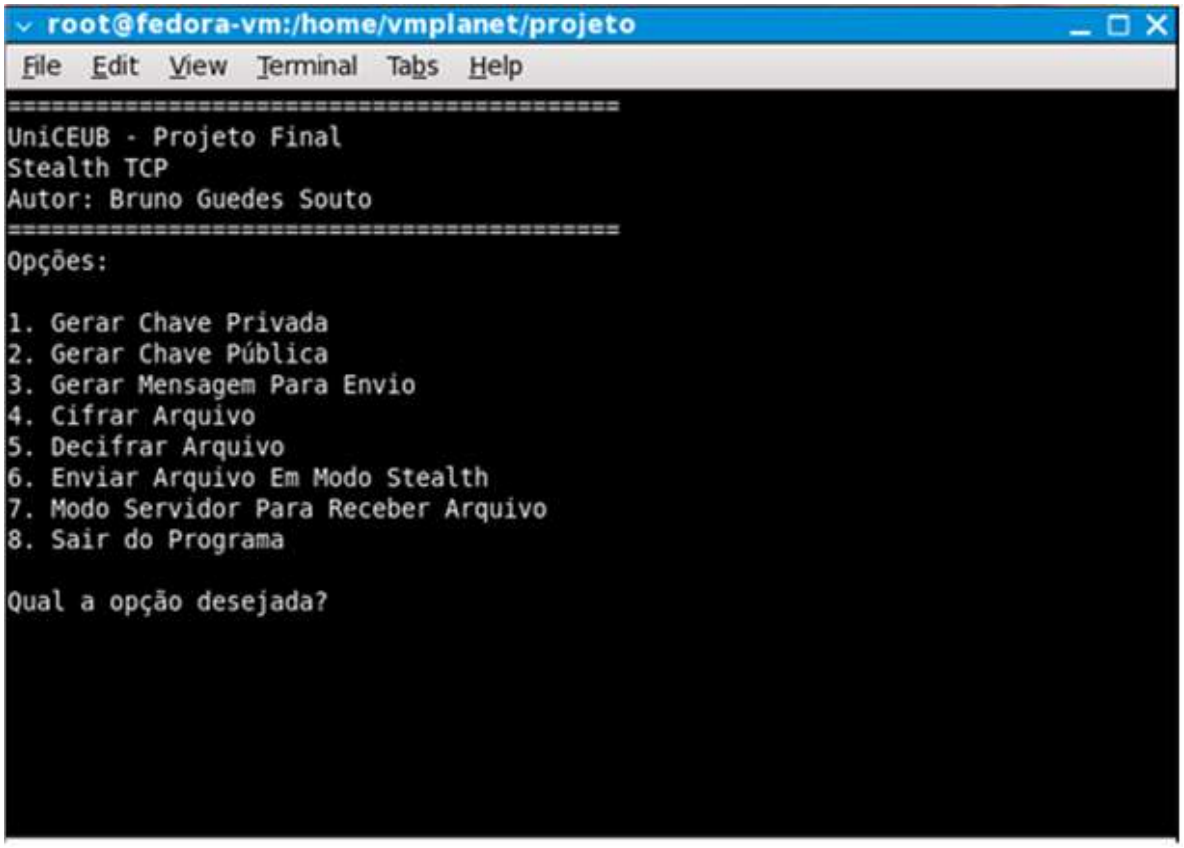

Figura 4.3 - Menu Interativo

O menu foi escrito utilizando linguagem de programação BASH, o código do menu é parte do resultado deste trabalho e encontra-se anexado no Apêndice 2 deste trabalho.

4.1.1. Geração do Par de Chaves

O computador servidor tem a opção de gerar uma chave pública ou privada a partir do menu. Para gerar uma chave pública o servidor deve ter ,obrigatoriamente, gerado uma chave privada. Pois a chave pública é gerada a partir da chave privada.

Ao escolher a opção para gerar a chave privada, o servidor deve escolher o nome do arquivo em que deseja armazenar a chave, conforme mostrado nas Figuras 4.3 e 4.4.

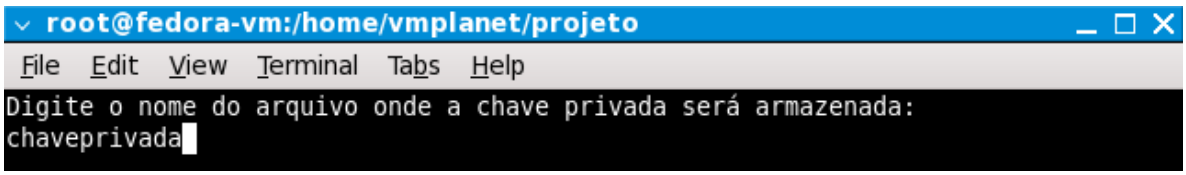

Figura 4.4 - Escolha do arquivo para armazenar a chave privada

Agora o servidor já possui uma chave privada que será utilizada para decriptografar os dados recebidos.

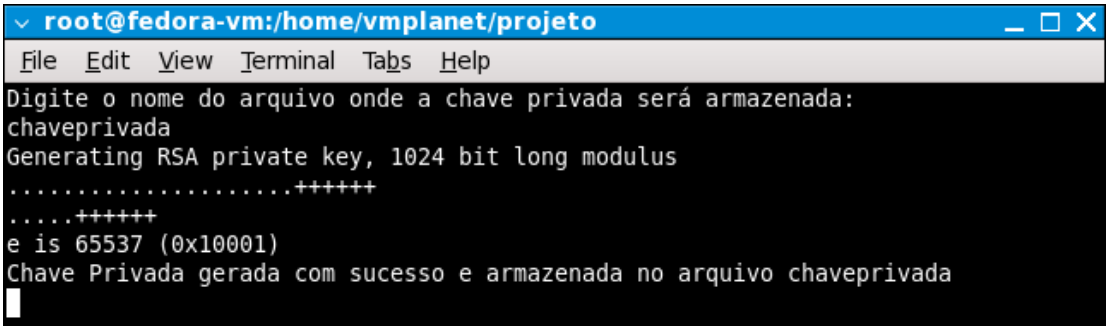

Figura 4.5 - Chave Privada Gerada com Sucesso

Agora o servidor deve gerar uma chave pública, que será utilizada para criptografar os dados que computadores clientes desejem enviar. A chave pública é gerada a partir da chave privada, para isso o servidor deve informar o nome do arquivo onde se encontra a chave privada, conforme mostrado nas Figura 4.5 e 4.6.

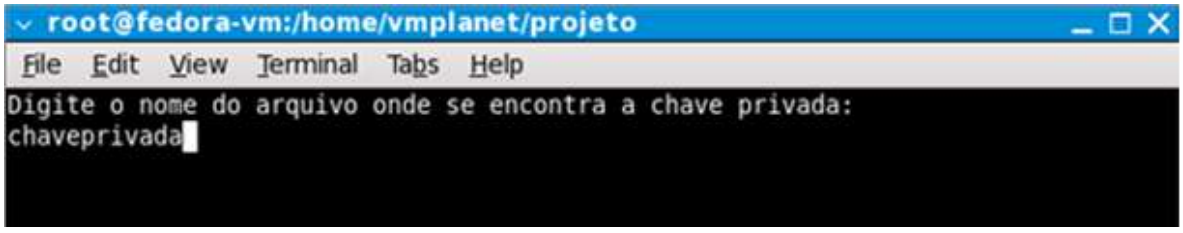

Figura 4.6 – Localização da chave privada para geração da chave pública

E então informar o nome do arquivo onde será armazenada a chave pública:

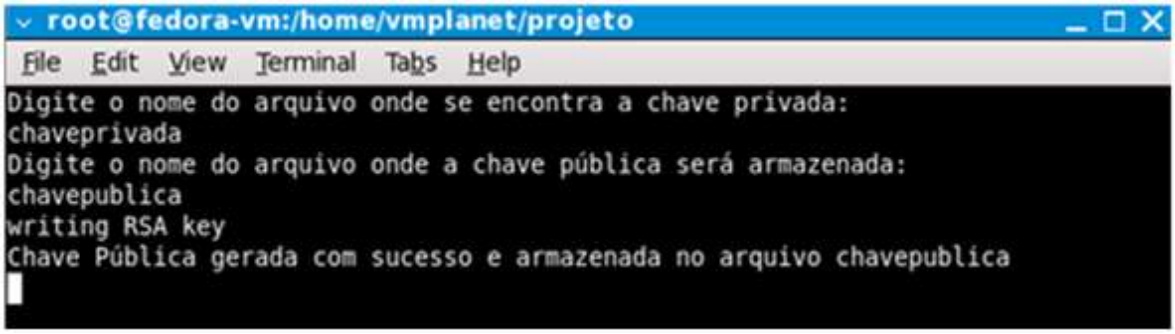

Figura 4.7 - Chave Pública Gerada com Sucesso

Desta maneira, o computador servidor pode distribuir a chave pública por vários meios digitais como correio eletrônico, sítios na internet, etc.. Desta maneira propiciando que vários computadores enviem mensagem a apenas um único computador servidor.

4.1.2. Modo Servidor para receber arquivo

O computador servidor agora tem a opção ficar em modo de espera para aguardar o recebimento de uma mensagem de um computador cliente.

Quando escolhido o modo servidor o primeiro passo é informar o nome do arquivo onde a mensagem criptografada será armazenada, conforme pode ser visto na figura 4.8:

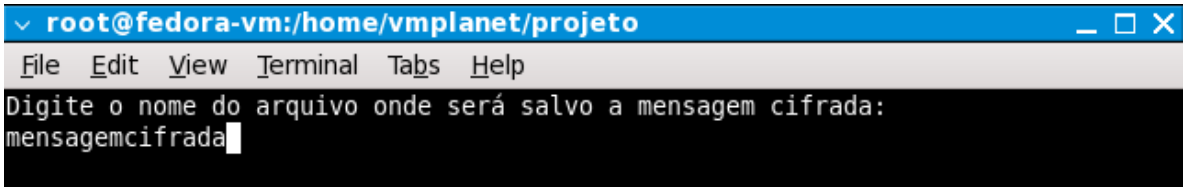

Figura 4.8 - Arquivo onde a mensagem cifrada será armazenada no servidor

Em seguida deve ser informado o endereço IP de onde se espera receber a mensagem como demonstra a figura 4.9

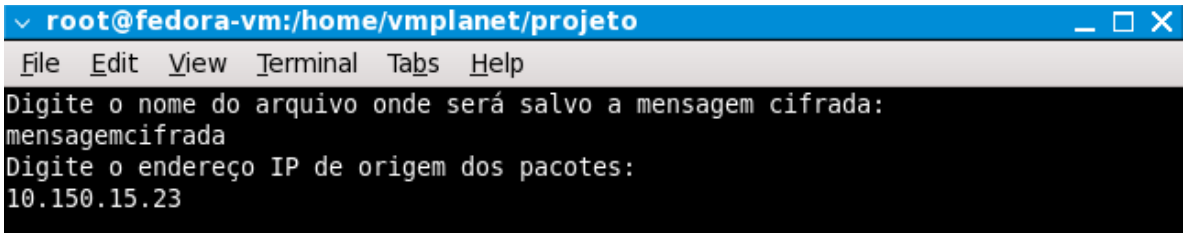

Figura 4.9 - Endereço IP de origem da mensagem

E finalmente deve ser informado a porta na qual se espera receber os dados de um computador cliente

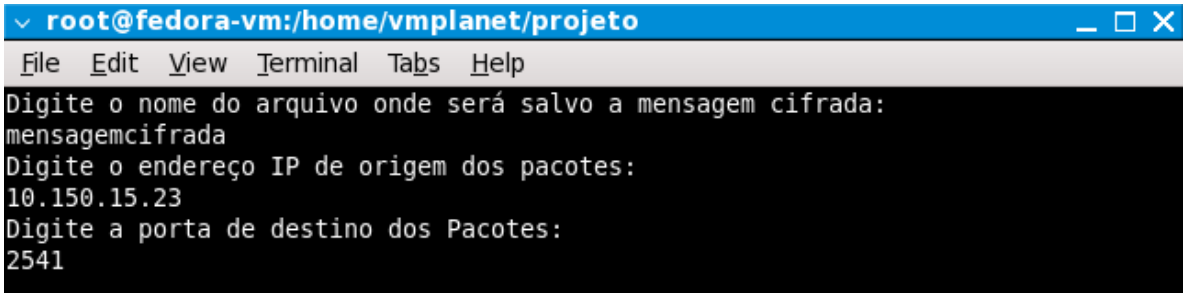

Figura 4.10 - Porta de destino da mensagem para escuta

Com estes dados informados, o aplicativo fica em modo de espera aguardando o recebimento da mensagem como pode ser visualizado na figura 4.11:

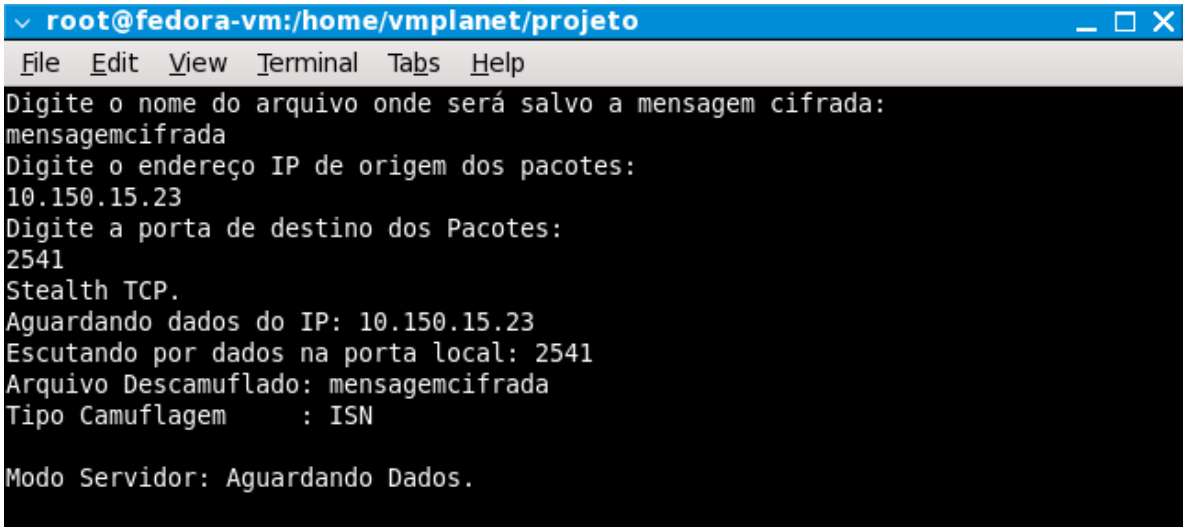

Figura 4.11 - Modo servidor aguardando dados.

4.1.3. Decriptografia

Neste passo o servidor irá decriptografar os dados que foram recebidos

Depois de ter recebido o arquivo com sucesso o computador servidor seleciona a opção de número 5 para realizar a decriptografia do arquivo para ter o seu conteúdo lido.

Como mostra a figura 4.12 primeiro deve ser informado o arquivo que contém a chave privada e logo depois informar o arquivo onde será armazenada a mensagem em texto claro:

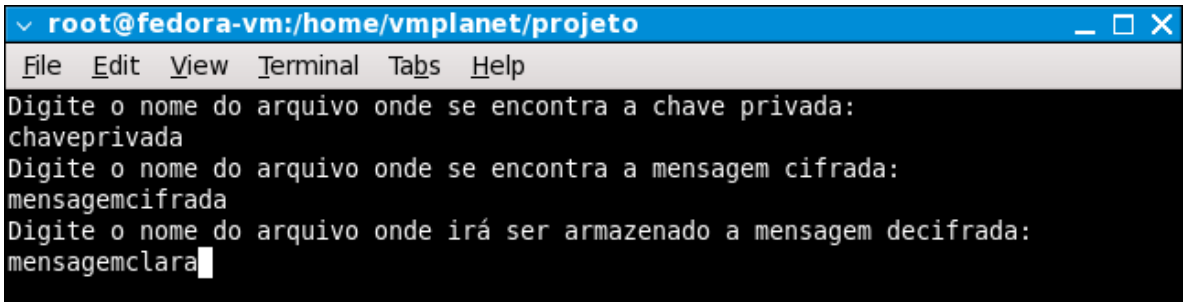

Figura 4.12 - Informação do arquivo de mensagem cifrada e mensagem clara

Finalmente o arquivo é decifrado utilizando a chave privada:

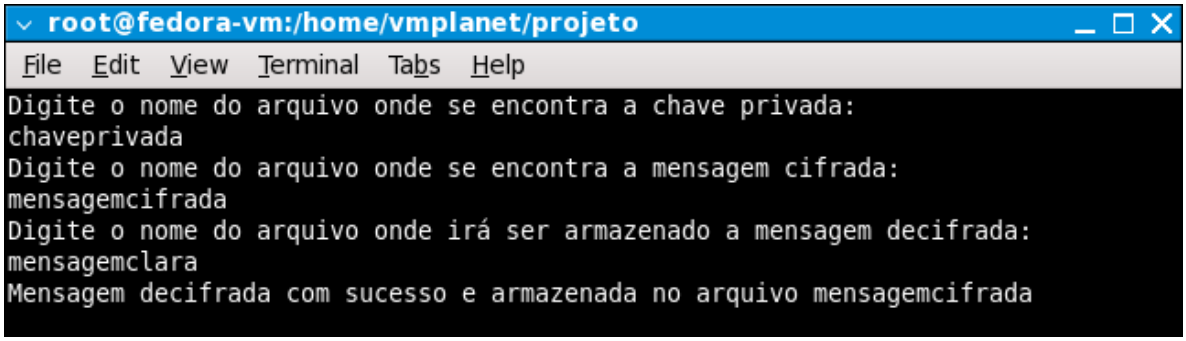

Figura 4.13 - Arquivo decifrado com sucesso

4.2. – Cliente

No computador representado como cliente na topologia serão realizadas três atividades. Primeiro, irá ser criado um arquivo de mensagem para ser transmitido a um computador servidor, em seguida o arquivo será criptografado para ser enviado com segurança a um computador servidor e finalmente o arquivo será enviado ao seu destino.

4.2.1. – Entrada de dados

O computador cliente após estar de posse da chave pública irá primeiramente acessar a opção de número 3 que possui a opção de gerar a mensagem que deverá ser enviada para o computador servidor.

O cliente deve digitar o nome do arquivo onde a mensagem deve ser armazenada.

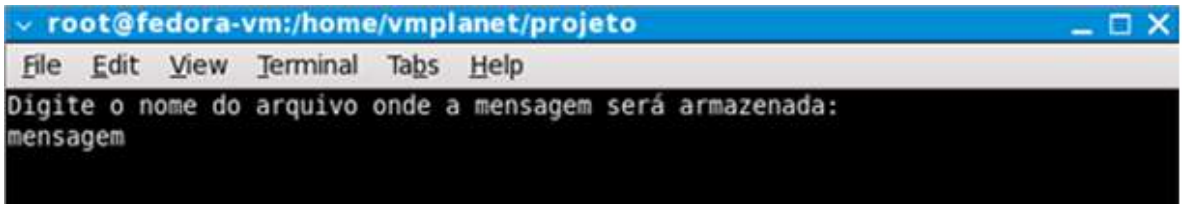

Figura 4.14 - Criação do arquivo para armazenar mensagem para envio

E então digitar a mensagem que será inserida no arquivo e enviada ao computador servidor. O processo é mostrado nas Figuras 4.14 e 4.15.

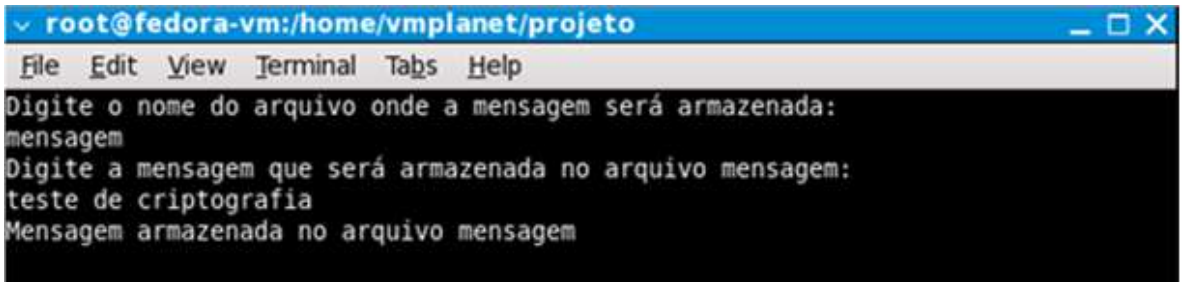

Figura 4.15 - Geração da Mensagem com Suceso

4.2.2. – Criptografia

Neste passo o cliente irá criptografar os dados para serem enviados para o computador servidor.

Ao selecionar a opção de número quatro o computador cliente tem a opção de cifrar o arquivo que será enviado ao computador servidor. Após seleciada a opção será necessário especificar qual chave pública deverá ser utilizada, como mostrado na Figura 4.16.

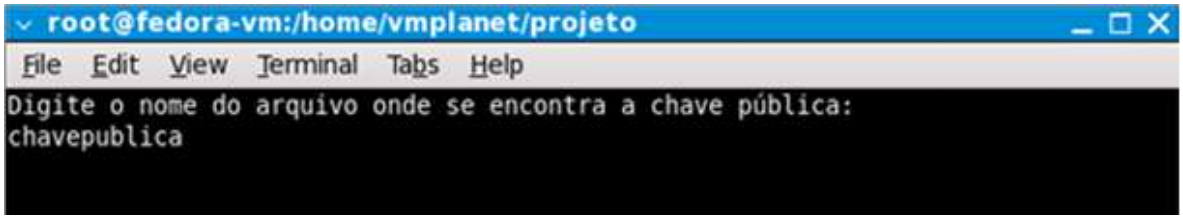

Figura 4.16 - Localização da chave pública para cifragem da mensagem

Selecionar o arquivo de mensagem que deverá ser cifrado

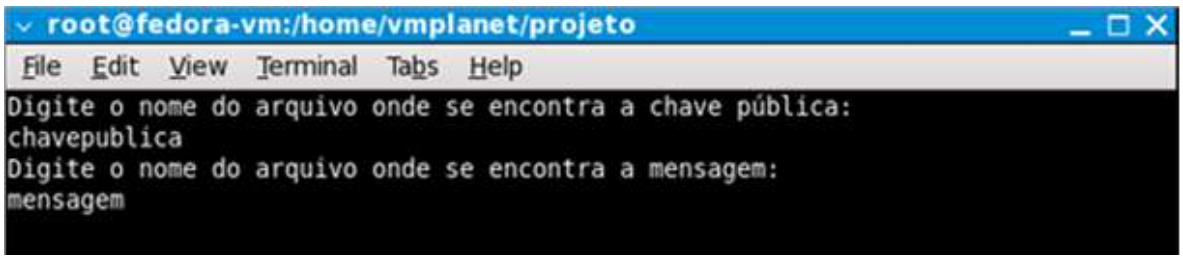

Figura 4.17 - Localização da mensagem a ser cifrada

Inserir o nome do arquivo onde deverá ser armazenada a mensagem cifrada. O processo é mostrado nas Figuras 4.16 e 4.17.

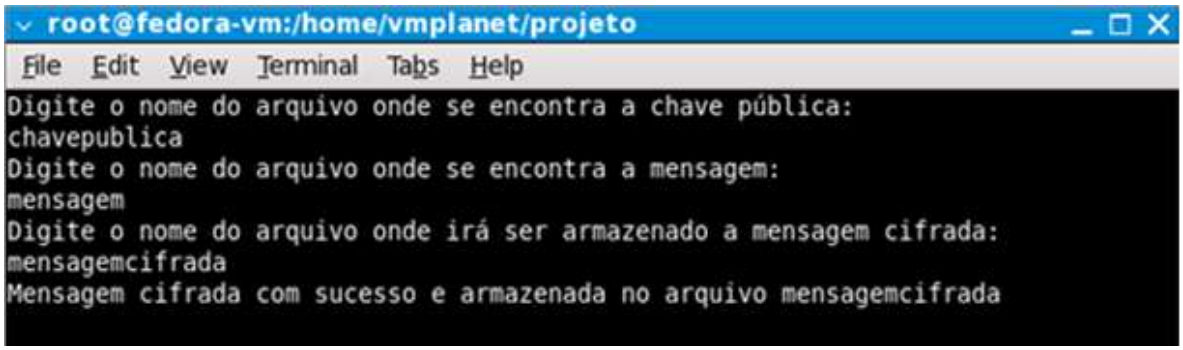

Figura 4.18 - Mensagem Cifrada com Sucesso

4.2.3. – Envio dos pacotes ao servidor

Na etapa de envio o computador cliente irá informar várias informações que serão utilizadas para construir o cabeçalho TCP/IP para envio dos pacotes ao computador servidor.

Primeiro o cliente informa em qual o arquivo está armazenada a mensagem cifrada, conforme mostra a Figura 4.19.

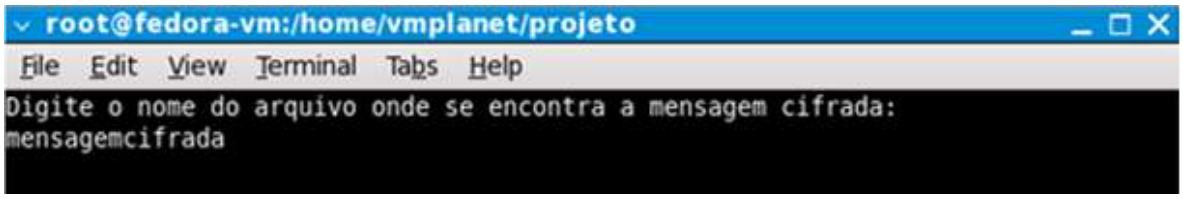

Figura 4.19 - Localização do arquivo cifrado para envio

Agora deve ser informado o endereço IP do computador servidor.

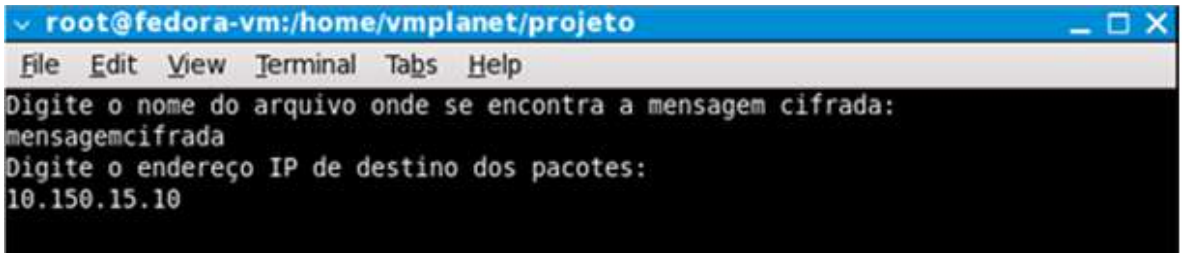

Figura 4.20 - Informação do endereço IP de Origem dos Pacotes

O endereço IP de origem pode ser o endereço real do computador cliente ou então um endereço forjado.

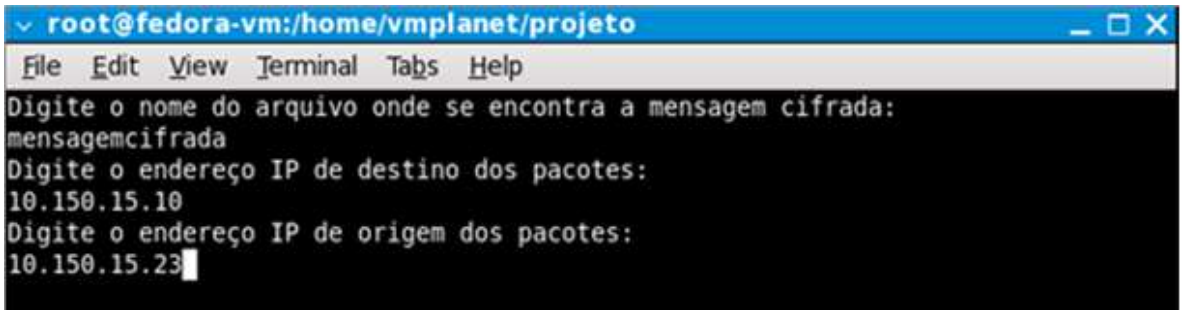

Figura 4.21 - Informação do endereço IP de Destino dos Pacotes

O cliente devera informar a porta de origem e a porta de destino dos pacotes.

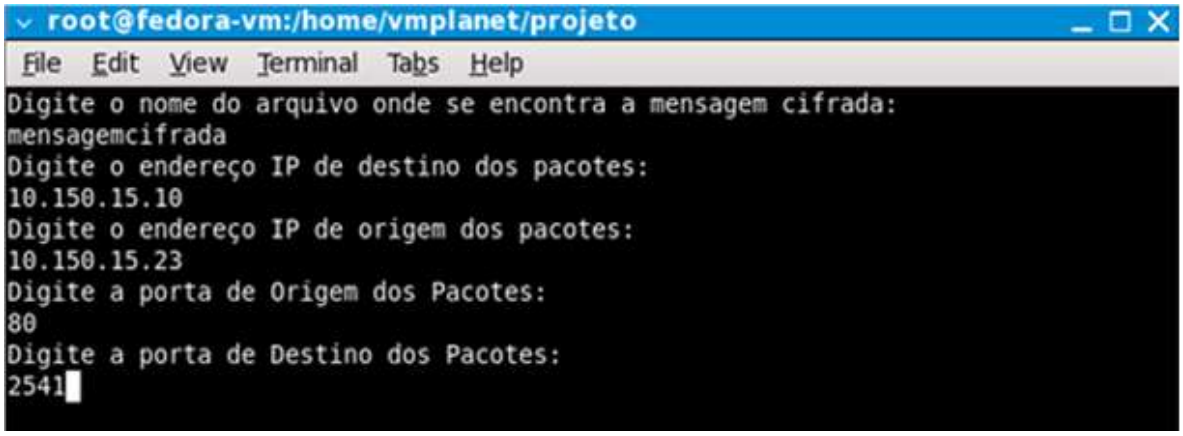

Figura 4.22 - Informação das portas de origem e destino dos pacotes.

E finalmente informar o delay em micro segundos que os pacotes deverão ser enviados. O delay neste caso é o tempo entre cada byte que será enviado para o servidor. Cada byte é o equivalente a um caractere da mensagem. As figuras de 4.19 a 4.24 mostram a seqüência do processo.

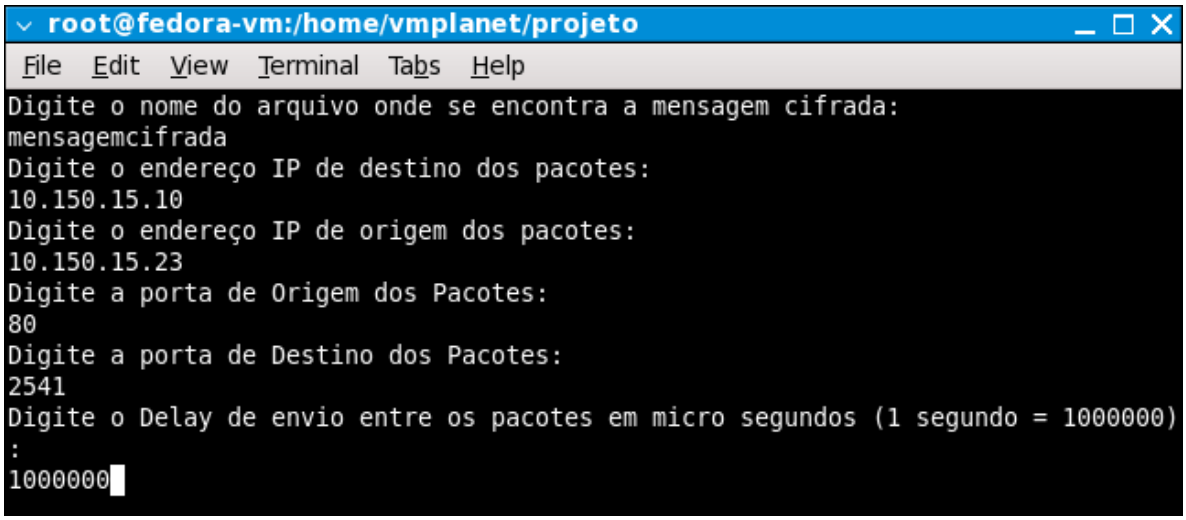

Figura 4.23 - Informação do Delay entre os Pacotes

E finalmente a mensagem começa a ser transmitida para o servidor como mostra a figura 4.24.

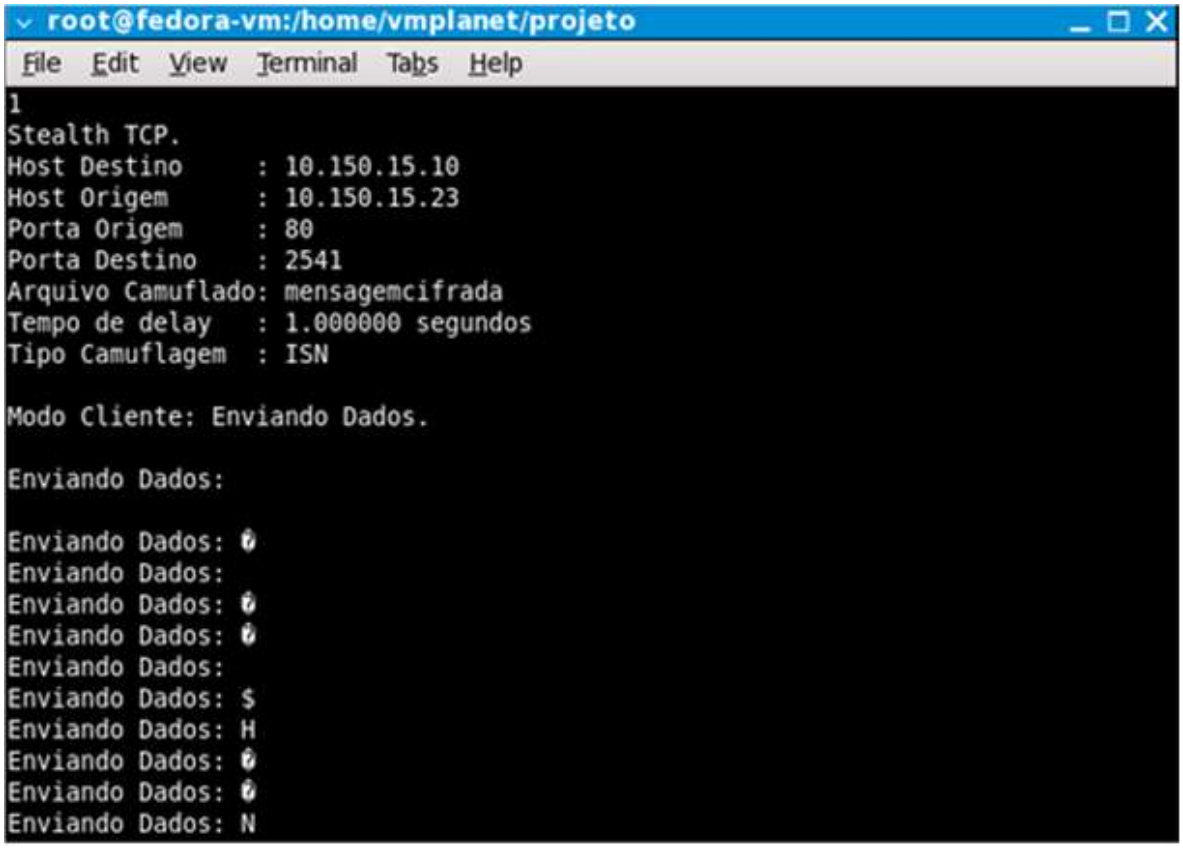

Figura 4.24 - Mensagem Sendo Enviada ao Computador Servidor

4.3. – Computador Atacante

O computador atacante tem apenas a função de tentar realizar a captura dos pacotes enviados para tentar reconstruir desvendar o conteúdo da mensagem, para captura dos pacotes foi utilizado o software WireShark em sua versão 1.0.4 e o resultado das capturas e testes podem ser consultados pelo leitor no capítulo 6 deste trabalho.

5. CAPÍTULO 5 – IMPLEMENTAÇÃO

Este capítulo explica e detalha as principais funcionalidades implementadas neste projeto, envolvendo a parte de software e hardware, assim como as configurações realizadas.

5.1. Software

O Software principal utilizado neste projeto foi desenvolvido em linguagem C e utilizado em ambiente Linux, o código principal é parte do apêndice 1 deste trabalho.

Num primeiro momento foi cogitado trabalhar com C# ou Java, em ambiente Windows, por ser um ambiente mais abrangente e linguagens que estão mais em voga atualmente.

Primeiramente esbarrou-se no problema do sistema operacional. O Windows XP a partir do *release* do Service Pack 2 (SP2) não suporta mais o envio de dados TCP utilizando *Raw Sockets* (Microsoft et al, 2007) . Isso invalida o uso

do Windows no projeto, a menos que se utilize versão anterior ao SP2, pois o objetivo é justamente enviar pacotes utilizando *raw sockets* no cabeçalho TCP.

Definido o Ambiente como Linux, que permite a manipulação de dados do cabeçalho TCP para envio, foi pensado na utilização da linguagem Java, por estar altamente em voga no mercado e ser uma linguagem robusta, fácil de programar e com material para pesquisa em alta disponibilidade tanto em livros, quanto na internet.

Infelizmente o Java não oferece suporte nativo à *Raw Sockets*, para isso deveria ser utilizada uma *Application Programming Interface* (API). Atualmente existem várias e para isto deveria ser feito o teste com algumas e este tempo de escolher uma API não era um luxo que estava disponível.

Para isso foi pensado no uso de uma linguagem também robusta, que oferecesse amplo material de pesquisa e ajuda e que oferecesse suporte a *raw sockets* nativamente. E, então, foi escolhida a linguagem C.

Como o objetivo do trabalho não é desenvolver novos métodos de criptografia e sim utilizar o algoritmo RSA, foi escolhido o *toolkit* OpenSSL que é um grupo de programas e rotinas em código aberto que implementa os protocolos *Secure Sockets Layer* (SSL) e *Transport Layer Security* (TLS) além de contar com

uma biblioteca de criptografia bastante extensa que é a parte utilizada neste trabalho.

Para tornar a interação dos programas mais fácil com o usuário foi desenvolvido um menu em bash script que realiza as principais operações de maneira automática e transparente para o usuário. O código deste menu é parte do apêndice 2 deste trabalho.

5.1.1. Funcionalidades implementadas

O software conta com 3 funcionalidades principais implementadas. As duas primeiras são efetuadas utilizando o OpenSSL Toolkit e geram o par de chaves e a criptografia e decriptografia dos dados. A terceira e principal funcionalidade é a de utilizar o cabeçalho TCP para esteganografar os dados criptografados. Para isto foi utilizado o software em linguagem C e os dados criptografados foram escondidos no campo ISN do cabeçalho TCP.

Para a geração das chaves, estamos utilizando a geração de chaves com 1024bits. No software o comando utilizado para essa geração é sempre chamando o módulo de gerar chaves RSA do OpenSSL, isto é feito da seguinte maneira:

openssl genrsa -out "nome do arquivo" 1024

Onde o argumento genrsa informa ao OpenSSL que queremos criar uma chave privada RSA, o argumento –out diz para que arquivo iremos salvar esta chave e o 1024 informa que queremos uma chave de 1024 bits.

A partir desta chave privada podemos gerar a chave pública com o comando:

openssl rsa -in "arquivo chave privada" -pubout > "arquivo chave publica"

Além da funcionalidade de gerar as chaves, o OpenSSL também é utilizado no projeto para realizar a criptografia e decriptografia dos dados.

Para a criptografia sempre utilizamos a chave pública e pedimos para criptografar o arquivo onde se encontra a mensagem que queremos enviar ao computador servidor. O comando utilizado é

openssl rsautl -encrypt -in "arquivo de mensagem" -pubin -inkey "arquivo de chave privada" -out "arquivo criptografado"

E a operação reversa é feita de maneira similar utilizando o comando

openssl rsautl -decrypt -in “arquivo criptografado” -inkey “arquivo de chave privada” -out “arquivo decriptografado”

Com o uso do OpenSSL podemos gerar chaves e criptografar os dados de maneira rápida, direta e confiável, mas a principal funcionalidade do projeto é a de esteganofragar os dados no cabeçalho TCP, especificamente no campo do ISN.

Para fazer isto a principal função do programa escrito em C é a função pacotefalso. A função pacote falso tem o seguinte protótipo:

void pacotefalso(unsigned int ip_origem, unsigned int ip_destino, unsigned short porta_origem, unsigned short porta_destino, char *filename, int server)

Dentro dessa função definimos três estruturas, que são:

```
struct envia_tcp
  {
     struct iphdr ip;
     struct tcphdr tcp;
  } envia_tcp;

struct recb_tcp
  {
     struct iphdr ip;
     struct tcphdr tcp;
     char buffer[10000];
```

```
  } recb_pct;
```

E

```
struct pseudo_header
  {
    unsigned int endereco_origem;
    unsigned int endereco_destino;
    unsigned char placeholder;
    unsigned char protocolo;
    unsigned short tcp_length;
    struct tcphdr tcp;
  } pseudo_header;
```

A estrutura envia_tcp é a estrutura usada para montar o cabeçalho TCP/IP e que é enviado para o computador servidor. A estrutura recb_tcp é a estrutura que será utilizada para o servidor para remontar as informações necessárias do cabeçalho TCP e, finalmente, temos a estrutura pseudo_header que é a estrutura necessária para calcular o checksum correto do cabeçalho TCP (KOZIEROK et al, 2005) .

No modo cliente, a primeira coisa a ser feita é ler os caracteres da mensagem armazenada no arquivo, um a um, utilizando a função fgetc() até o final do arquivo. A função fgetc() lê um caractere do arquivo e retorna o seu valor em formato de um número inteiro.

Assim temos:

```
if(server==0)
{
if((input=fopen(filename,"rb"))== NULL)
{
printf("Impossivel abrir o arquivo %s para leitura\n",filename);
exit(1);
}
else while((ch=fgetc(input)) !=EOF)
{
```

O programa testa para ver se estamos no modo cliente e então tenta ler o arquivo e armazenar os caracteres na variável ch.

No momento de geração das informações do cabeçalho TCP é importante lembrar que geramos um pacote para cada byte, logo um pacote para cada caractere. Em outras palavras a função pacotefalso() será chamada várias vezes, uma para cada caractere. Assim, inserimos no campo do cabeçalho TCP a informação armazenada na variável ch dentro do campo ISN. Fazemos esta atribuição utilizando:

```
envia_tcp.tcp.seq = ch;
```

Onde envia_tcp é a estrutura previamente definida e seq é o campo equivalente do cabeçalho TCP ao ISN.

Os outros campos do cabeçalho TCP/IP são construídos com valores padrões ou randômicos gerados pelo software. Os únicos campos que são setados pelo usuário são as portas de origem e destino e os endereços de origem e destino que são lidos como argumentos do comando.

```
envia_tcp.ip.saddr = ip_origem;
envia_tcp.ip.daddr = ip_destino;
envia_tcp.tcp.source = htons(porta_origem);
envia_tcp.tcp.dest = htons(porta_destino);
```

Finalmente, o programa abre o socket cru para envio ao computador servidor,

```
envia_socket = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
```

e envia o pacote com o as informações necessárias

```
sendto(envia_socket, &envia_tcp, 40, 0, (struct sockaddr *)&sin, sizeof(sin));
```

e mostra na tela os caracteres que estão sendo enviados.

```
printf("Enviando Dados: %c\n",ch);
```

No modo servidor criamos um arquivo para leitura e escrita:

```
output=fopen(filename,"wb")
```

E abrimos um socket para leitura dos pacotes

```
recb_socket = socket(AF_INET, SOCK_RAW, 6);
```

Então colocamos o servidor em modo de escuta

```
read(recb_socket, (struct recb_tcp *)&recb_pct, 9999);
```

Se o pacote tiver a flag SYN/ACK setada e for do endereço correto, o servidor captura o pacote, mostra o caractere enviado na tela e armazena o caractere no arquivo especificado como argumento.

```
if (porta_origem == 0)
{
        if((recb_pct.tcp.syn == 1) && (recb_pct.ip.saddr == ip_origem))
        {
                /* Decode do ISN */
                printf("Recebendo Dados: %c\n",recb_pct.tcp.seq);
                fprintf(output,"%c",recb_pct.tcp.seq);
                fflush(output);
        }
}
```

O comando fprintf traduz o inteiro gerado pelo comando fgetc de volta para um caractere e o armazena no arquivo desejado.

Mesmo que o endereço de origem esteja errado, mas os pacotes estejam destinados à porta certa, o servidor também captura o pacote, mostra o caractere enviado na tela e armazena o caractere no arquivo especificado como argumento.

```
else
    {
        if((recb_pct.tcp.syn==1) && (ntohs(recb_pct.tcp.dest) == porta_origem))
        {
            printf("Recebendo Dados: %c\n",recb_pct.tcp.seq);
            fprintf(output,"%c",recb_pct.tcp.seq);
            fflush(output);
        }
    }
```

o socket finalmente é fechado

```
close(recb_socket);
```

e o arquivo finalizado

```
fclose(output);
```

Com isso agora se tem um arquivo com os caracteres recebidos pelo servidor que foram enviados pelo cliente.

6. CAPÍTULO 6 – TESTES

Neste capítulo iremos descrever os três testes realizados com o software e seus resultados. Foram efetuados três testes sendo eles, um teste de rede, um teste de confiabilidade e um teste de confidencialidade.

6.1. – Testes de Rede

Nos testes de rede, o objetivo foi medir o impacto dos pacotes na rede para saber se o envio deles poderia honerar a performance da rede de alguma maneira e também checar o quão rápido poderíamos transmitir os pacotes pela rede garantindo a sua chegada do outro lado caso a rede estivesse sobrecarregada.

Primeiramente foi medida a vazão da rede utilizada entre o computador servidor e o computador cliente. A vazão foi medida utilizando o programa PCAUSA Test TCP Utility. E foi medida uma vazão de aproximadamente 88Mbits, ou cerca de 11275 KB/seg como pode ser visto na figura 6.1.

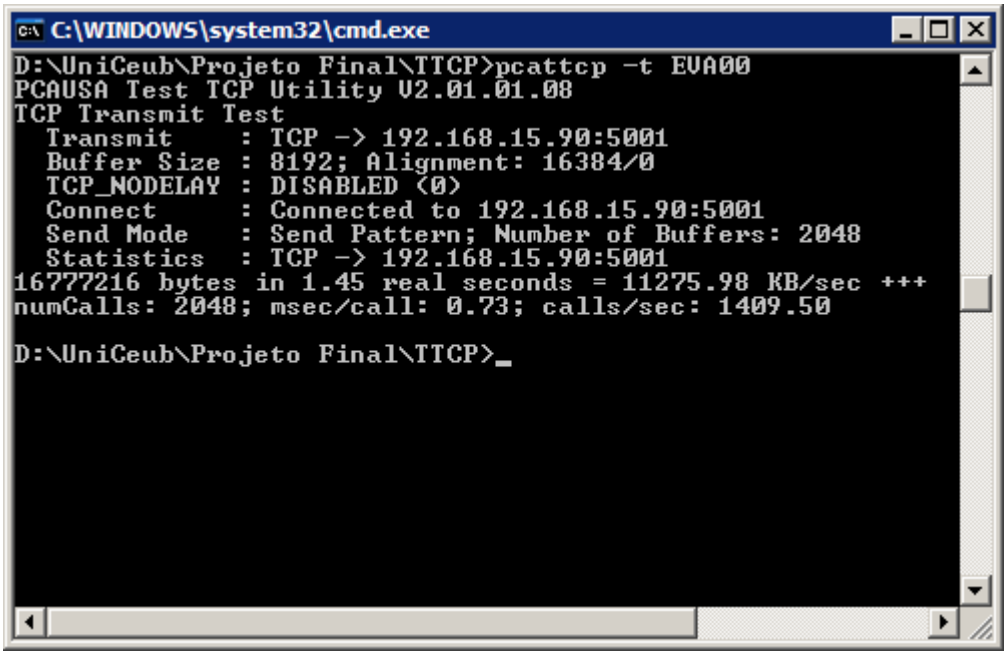

Figura 6.1 - Medição da vazão utilizando o programa TTCP

Os pacotes gerados pelo software tem um tamanho de 60 bytes, como pode ser observado na figura 6.2 durante a captura pelo software WireShark.

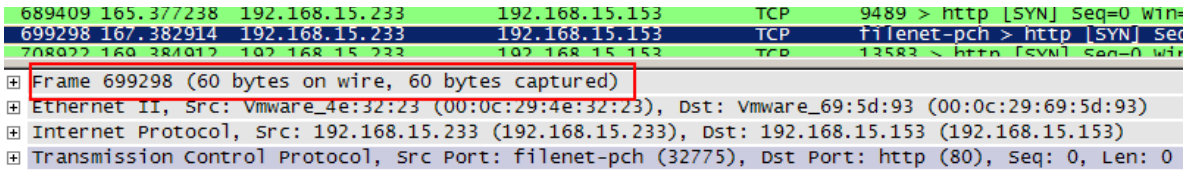

Figura 6.2 - Tamanho do pacote gerado pelo software

Com isso, podemos chegar à conclusão que impacto dos pacotes na rede testada com vazão de 88mbits é mínimo. Pois as mensagens geradas tem um

tamanho 128 caracteres, como enviamos um caractere de cada vez, isso significa que enviamos 128 pacotes. Desta maneira mesmo que enviássemos a mensagem da maneira mais rápida possível estaríamos enviando um total de 7,5KB de informação, por uma rede que pode trafegar cerca de 1500 vezes esse volume por segundo.

O segundo teste envolveu sobrecarregar a rede para efetuarmos o envio dos pacotes pelo computador cliente e servidor.

Para sobrecarregar a rede foi utilizado um arquivo de uma imagem de um disco de DVD com cerca de 4.37GB. Para garantir o máximo uso da rede o arquivo foi duplamente enviado, tanto pelo cliente para o servidor, como do servidor para o cliente.

A partir do momento que começava a transmissão do arquivo entre os computadores foi iniciada a transmissão de uma mensagem pelo software. Neste software podemos utilizar a opção de delay, ou seja, podemos enviar cada pacote separado por um período de tempo. Para efeitos dos testes utilizamos os delays de 2, 1, 0.5, 0.1, 0.01, 0.001 e 0.001 segundos. Desta maneira conseguimos os seguintes resultados mostrados na tabela 6.1 e na figura 6.3

Tabela 6.1 - Arquivos enviados e recebidos com sucesso

Delay (em segundos)	Arquivos Enviados	Arquivos Recebidos com Sucesso
2	10	10
1	10	10
0.5	10	10
0.1	10	10
0.01	10	7
0.001	10	0
0.0001	10	0

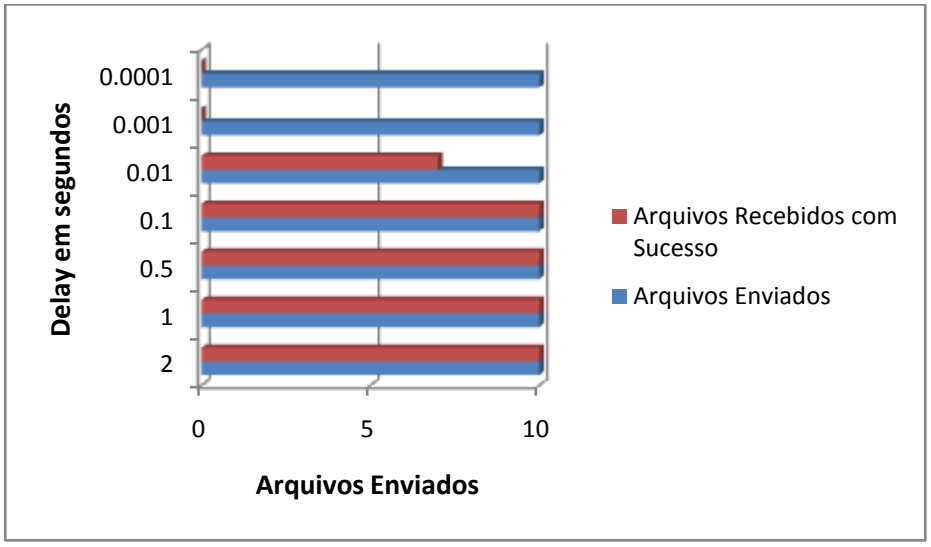

Figura 6.3 - Gráfico dos arquivos recebidos com sucesso

Para definirmos se os arquivos foram recebidos com sucesso ou não foi sempre calculado a soma md5. A soma md5 calcula um hash do arquivo como se fosse uma impressão digital. Logo qualquer alteração deste arquivo resulta num resultado diferente da soma md5. Depois de realizado os testes foram calculados as somas md5 de todos os arquivos e podemos comprovar que usando delays abaixo de 0.01 segundos os arquivos começam a ter problemas para serem

enviados corretamente e acabam se perdendo em algum ponto da rede. Como o objetivo deste trabalho não é o de transferir os arquivos da maneira mais rápida possível mas sim de transmiti-los da maneira mais segura, o resultado dos testes não causa nenhum impacto negativo no software.

6.2. – Testes de Confiabilidade

Nos testes de confiabilidade o objetivo era testar a dificuldade de se identificar os nossos pacotes numa rede carregada. Para este teste efetuamos a mesma transferência de arquivos entre o servidor e o cliente descrita no item 6.1 e utilizamos o envio dos pacotes com um delay de dois segundos para garantir que todos seriam entregues.

Durante o envio dos pacotes efetuamos a captura do tráfego da rede utilizando um computador atacante que capturava o tráfego utilizando o software WireShark.

Durante os cerca de quatro minutos e meio de transmissão foram capturados um total de 918944 pacotes na rede. De uma maneira que nosso tráfego de 128 pacotes durante esse período corresponde 0,014% do tráfego total

transmitido. Temos que levar em conta também que temos uma média de 3403 pacotes por segundo e estamos transmitindo apenas um pacote do nosso software a cada dois segundos.

Desta maneira, pode-se chegar à conclusão que a visualização dos nossos pacotes na rede e identificação dos mesmos é muito difícil. Um atacante que não estivesse procurando especificamente pelos pacotes gerados por nosso software teria uma grande dificuldade em identificá-los pois os mesmos se misturam com o tráfego normal da rede de maneira satisfatória. Temos que levar em conta também que utilizamos um delay de 2 segundos, poderia ser utilizado um delay ainda maior e com isso tornar a identificação destes pacotes ainda mais complicada.

Vale ressaltar que o ambiente em que os testes foram realizados é um ambiente ideal, onde a captura é desejada para efeitos de testes. Devido a segurança de redes se tornar cada dia uma missão mais critica tanto para empresas como usuários comuns a captura destes dados se torna também cada vez mais complicada com a utilização de switches e roteadores mais avançados tecnologicamente.

O resultado da captura destes pacotes pode ser encontrada no DVD em anexo com este trabalho e pode ser visualizada tanto com os programas WireShark como com o TCPDUMP.

6.3. - Teste de confidencialidade

O objetivo do teste de confidencialidade é comprovar que mesmo que a confiabilidade dos pacotes fosse quebrada, ou seja, que um atacante identificasse o conteúdo sendo transmitido pela rede. Ele não seria capaz de recuperar a informação transmitida num formato compreensível.

Usando mais uma vez o software WireShark podemos visualizar o conteúdo armazenado no cabeçalho TCP no campo do ISN. Supondo que nosso tráfego tenha sido identificado que o computador atacante tivesse filtrado nossos pacotes de maneira a identificar todo o conteúdo transmitido. O que ele visualizaria no campo do ISN seriam apenas os dados criptografados.

Pode-se ver isso claramente com as figuras 6.4 a 6.7. Na figura 6.4 temos um trecho do que foi transmitido para o computador servidor, marcado estão

caracteres que são de fácil reconhecimento no cabeçalho, são os caracteres P,\,A,H e U.

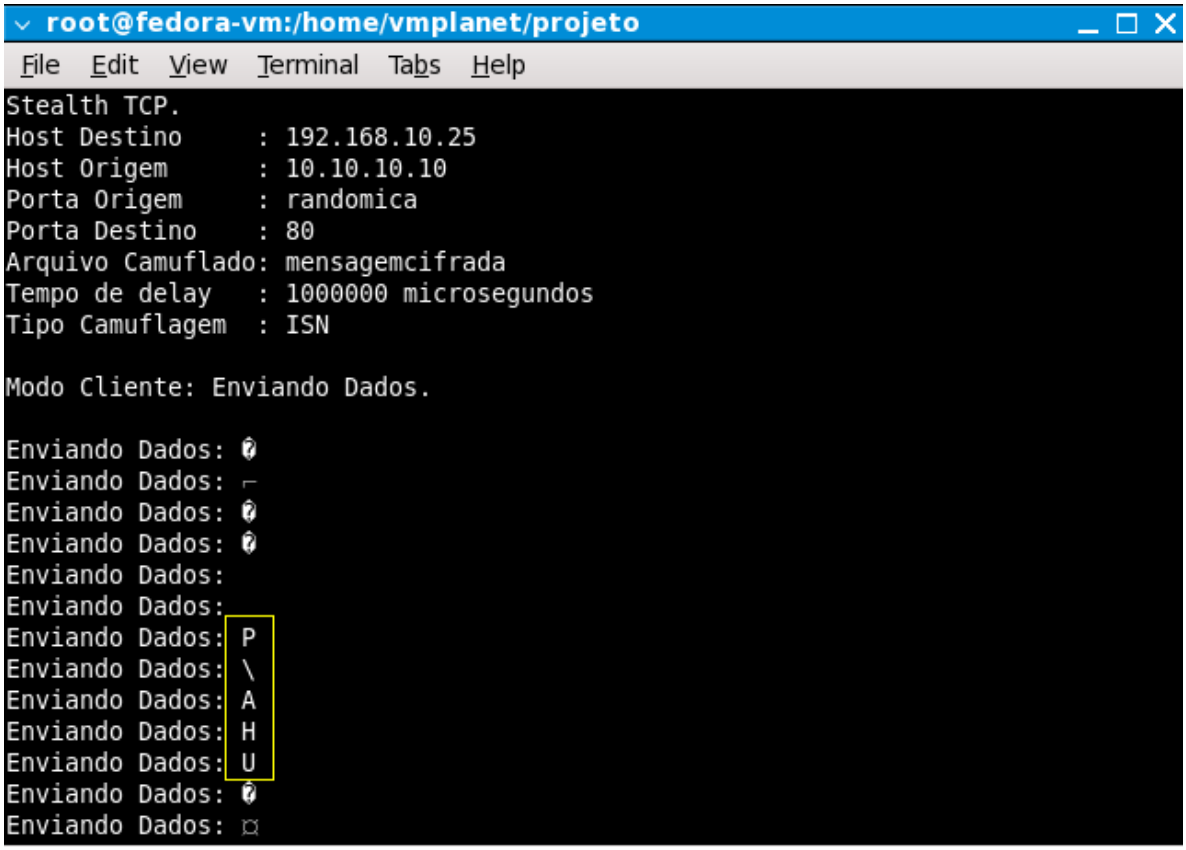

Figura 6.4 - Trecho de envio de mensagem ao servidor

Na figura 6.5 temos a captura efetuada desta transmissão pelo software WireShark:

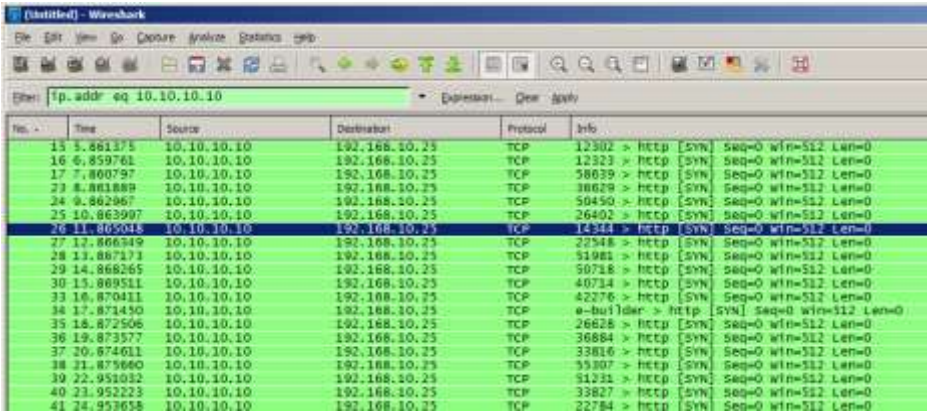

Figura 6.5 - Trecho de captura pelo software WireShark

A figura 6.6 mostra uma montagem da captura dos pacotes, e as informações do campo ISN, pode-se ver os caracteres de valor hexadecimal e também os caracteres enviados P,\,A,H e U, caso a confiabilidade dos pacotes fosse quebrada, o computador atacante poderia reconstruir todo o fluxo de dados e recuperar a mensagem enviada. Mas mesmo desta maneira ele teria apenas uma mensagem criptografada. A única maneira de o atacante recuperar esta mensagem seria utilizando a chave privada par da chave pública que foi utilizada para gerar esta mensagem.

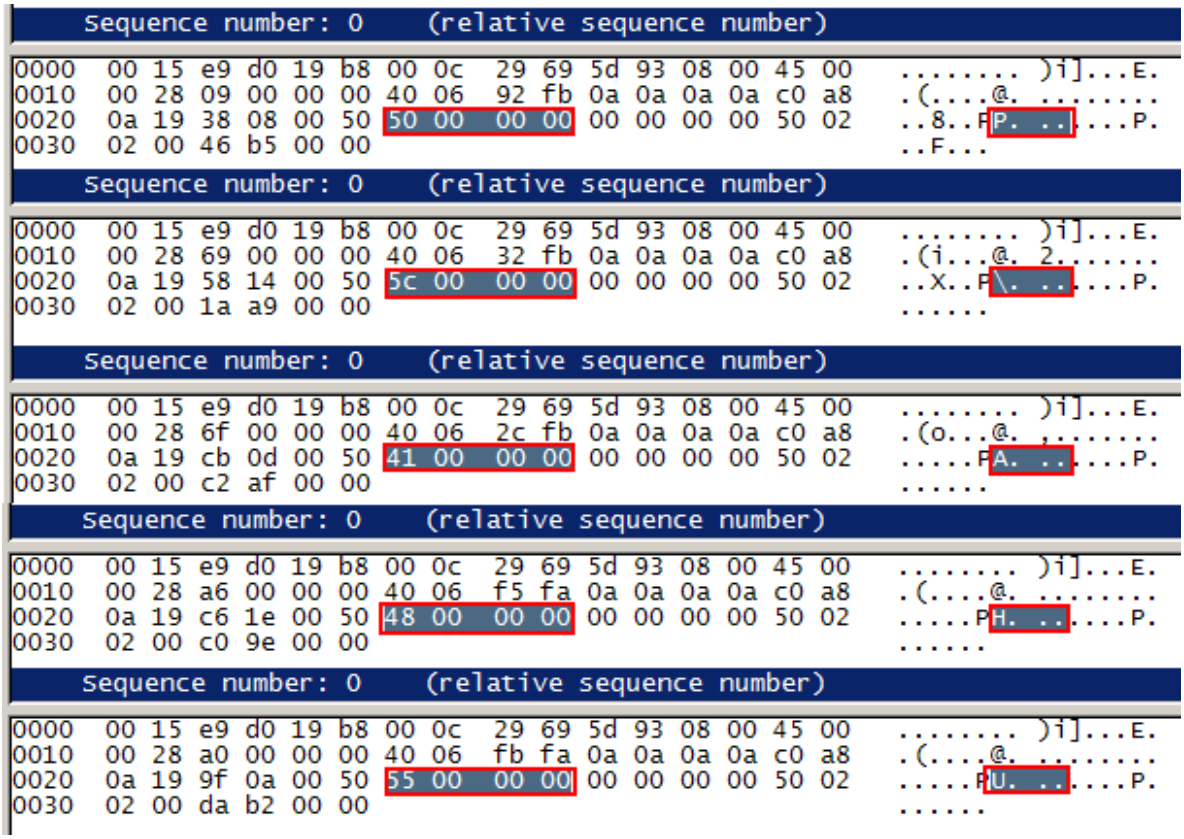

Figura 6.6 - Captura de Pacote e Visualização do Campo ISN

A chave utilizada pelo nosso software é de 1024 bits, até os dias de hoje uma chave de 1024 bits ainda não foi fatorada no mundo (http://www.rsa.com/rsalabs/node.asp?id=2092). O que torna até o presente momento as chaves de 1024 bits seguras. Desta maneira, podemos garantir a confidencialidade da mensagem mesmo que a sua confiabilidade seja quebrada e os pacotes identificados.

7. CAPÍTULO 7 – CONCLUSÃO

O software desenvolvido neste projeto e os resultados dos testes comprovam que o envio de dados esteganografados e criptografados utilizando o cabeçalho TCP/IP é possível e é seguro. Também conseguimos utilizar uma outra técnica de esteganografia de dados muito pouco explorada, quando a maioria das técnicas de esteganografia empregadas hoje são as utilizadas com imagens e vídeo. O resultado dos testes comprova que os pacotes enviados são de difícil detecção e mesmo quando detectados a mensagem não pode ser decifrada devido ao algoritmo de criptografia empregado.

As dificuldades encontradas durante o desenvolvimento do projeto foram solucionadas, sendo que em sua maioria foram as dificuldades com a manipulação direta do cabeçalho TCP. Devido as linguagens mais modernas em sua maioria não oferecem suporte nativo a essa funcionalidade e as restrições aplicadas nas versões mais recentes dos sistemas operacionais Windows.

Como continuidade ao desenvolvimento deste projeto podem ser implementadas melhorias para aumentar a confiabilidade dos pacotes assim como utilizar de outros meios de esteganografia para esconder os dados desejados. Podem ser aprimorados o sistema de delay para enviar os pacotes com delays

variados entre si, e utilizar outros campos do cabeçalho TCP/IP, ou até mesmo o uso de outro protocolo de rede para esconder os dados de maneira similar.

Outras implementações podem ser feitas mudando a maneira como o servidor identifica os pacotes que ele deve ler, e desta maneira gerar uma função que cria números randômicos do IP de origem baseados numa máscara de rede fornecida e como final a aplicação atual pode ser migrada para a próxima versão do protocolo IP, o IPv6.

8. REFERÊNCIAS BIBLIOGRÁFICAS

COMER, Douglas. Interligação de redes com TCP/IP Volume 1. Rio de Janeiro: Campus, 2006.

KATZ, Jonathan e LINDELL, Yehuda. *Introduction to Modern Cryptography*. Chapman & Hall, 2007.

KOZIEROK, Charles. *The TCP IP Guide*. Disponivel em: <http://www.tcpipguide.com/free/t_TCPChecksumCalculationandtheTCPPseudoHeader-2.htm>. Acesso em: 8 de novembro de 2008.

PARKER, Don e SUES, Mike. *Packet forensics using TCP*. Disponível em: <http://www.securityfocus.com/infocus/1845> Acesso em: 24 de outubro de 2008.

MICROSOFT. *Networking programs that send TCP packets or UDP packets over raw IP sockets may stop working after you apply security update MS05-019 to a computer that is running Windows XP with Service Pack 1*. Disponível em: <http://support.microsoft.com/kb/897656>. Acesso em: 10 de setembro de 2008.

SINGH, Simon. *The code book: the science of secrecy from ancient egypt to quantum cryptography*. Nova Iorque: Anchor Books, 1999.

STALLINGS, William. Criptografia e Segurança de Redes. São Paulo: Pearson Prentice Hall, 2008.

TANENBAUM, Andrew. *Computer Networks*. Prentice Hall, 2002.

WAYNER, Peter. *Disappearing cryptography: information hiding, steganography & watermarking*. Boston: AP Professional Books, 2002.

9. APÊNDICES

9.1. Apêndice 1 – Código fonte do programa principal

```
/* Stealth TCP

* Escrito e Adptado por Bruno Guedes Souto (brunoguedes@gmail.com)
*
* Este programa manipula o cabeçalho TCP/IP especificamente
* na porção do ISN pra enviar um arquivo um byte de cada vez
* para um host destino. Este programa funciona tanto como cliente
* e como servidor e pode ser usado para esconder a transmissão
* de dados dentro de um cabeçalho IP.
*
* Este Programa faz parte de tese para conclusão de curso de Bacharel em
* Engenharia da Computação pelo Centro Universitário de Brasília - UniCEUB
* e deve ser usado em conjunto com o documento desenvolvido nomeado:
* Envio de mensagens utilizando esteganografia em pacotes TCP/IP com
* cabeçalho criptografado com algorítmo RSA
*
* Este software deve ser utilizado por sua própria conta e risco.
*
* Este software não pode ser utilizado para fins comerciais sem autorização.
*
* Para compilar: gcc -o stealth_tcp stealth_tcp.c
*
* Neste código foram utilizadas porções ora modificadas ora em sua integra dos seguintes códigos:
* covert_tcp.c (c) 1996 Craig H. Rowland
* ping.c (c) 1987 Regents of the University of California. (See function in_cksm() for details)
* synhose.c
* outros códigos sem autoria específica encontrados na internet.
*/

#include <stdio.h>
```

```
#include <stdlib.h>
#include <signal.h>
#include <string.h>
#include <unistd.h>
#include <netdb.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <linux/ip.h>
#include <linux/tcp.h>

/* Protótipos */
void pacotefalso(unsigned int, unsigned int, unsigned short, unsigned
short,char *,int,unsigned int);

unsigned short in_cksum(unsigned short *, int);

unsigned int host_convert(char *);

void ajuda(char *);

main(int argc, char **argv)
{
   unsigned int host_origem=0,host_destino=0,delay=0;
   unsigned short porta_origem=0,porta_destino=80;
   int server=0,file=0;
   int count;
   char hostdestino[80],hostorigem[80],filename[80];

   /* Title */
   printf("Stealth TCP.\n");
     /* Teste de permissão */
   if(geteuid() !=0)
    {
    printf("\nVocê precisar ser root para rodar este programa.\n\n");
    exit(0);
    }
```

```
   /* Mostra ajuda */
   if((argc < 6) || (argc > 15))
   {
   ajuda(argv[0]);
   exit(0);
   }

/* Checa argumentos e setups iniciais */
   for(count=0; count < argc; ++count)
    {
    if (strcmp(argv[count],"-destino") == 0)
     {
     host_destino=host_convert(argv[count+1]);
     strncpy(hostdestino,argv[count+1],79);
     }
    else if (strcmp(argv[count],"-origem") == 0)
     {
     host_origem=host_convert(argv[count+1]);
     strncpy(hostorigem,argv[count+1],79);
     }
    else if (strcmp(argv[count],"-arquivo") == 0)
     {
     strncpy(filename,argv[count+1],79);
     file=1;
     }
    else if (strcmp(argv[count],"-porta_origem") == 0)
      porta_origem=atoi(argv[count+1]);

    else if (strcmp(argv[count],"-porta_destino") == 0)
      porta_destino=atoi(argv[count+1]);
    else if (strcmp(argv[count],"-server") == 0)
      server=1;
    else if (strcmp(argv[count],"-delay") == 0)
      delay=atoi(argv[count+1]);
    }
```

```
   /* Checa pelo nome do arquivo */


   if(file != 1)
    {
    printf("\n\nVoce precisa informar um nome de arquivo (-arquivo
<nomedoarquivo>)\n\n");
    exit(1);
    }


   if(delay == 0)
     delay=1000000; /* Seta o delay padrão como 1 segundo */




   if(server==0) /* Modo cliente */
    {
     if (host_origem == 0 && host_destino == 0)
      {
      printf("\n\nVoce precisa informar um IP de origem e destino para o
modo cliente.\n\n");
      exit(1);
      }
     else if (delay <= 0)
      {
      printf("\n\nVoce precisa informar um tempo de espera superior a 0
segundos.\n\n");
      exit(1);
      }
     else
      {
      printf("Host Destino     : %s\n",hostdestino);
      printf("Host Origem      : %s\n",hostorigem);
       if(porta_origem == 0)
        printf("Porta Origem     : randomica\n");
       else
```

```
        printf("Porta Origem     : %u\n",porta_origem);
       printf("Porta Destino    : %u\n",porta_destino);
       printf("Arquivo Camuflado: %s\n",filename);
       printf("Tempo de delay   : %u microsegundos\n",delay);
       printf("Tipo Camuflagem  : ISN\n");
       printf("\nModo Cliente: Enviando Dados.\n\n");
       }
      }

    else /* Modo Servidor */
     {
     if ( host_origem == 0 && porta_origem == 0)
      {
      printf("Voce precisa fornecer um endereco de origem e uma porta de
origem para o modo servidor.\n");
      exit(1);
      }

     if(host_destino == 0)
      strcpy(hostdestino,"Qualquer Host");
     if(host_origem == 0)
      strcpy(hostorigem,"Qualquer Host");
     printf("Aguardando dados do IP: %s\n",hostorigem);
     if(porta_origem == 0)
      printf("Escutando por dados na porta local: Qualquer Porta\n");

     else
      printf("Escutando por dados na porta local: %u\n",porta_origem);
      printf("Arquivo Descamuflado: %s\n",filename);
      printf("Tipo Camuflagem     : ISN\n");
      printf("\nModo Servidor: Aguardando Dados.\n\n");
     }

     /* Chama função principal */

     pacotefalso(host_origem, host_destino, porta_origem, porta_destino,
filename,server,delay);
```

```
exit(0);

}

void pacotefalso(unsigned int ip_origem, unsigned int ip_destino,
unsigned short porta_origem, unsigned short porta_destino, char
*filename, int server, unsigned int delay)
{
   struct envia_tcp
   {
      struct iphdr ip;
      struct tcphdr tcp;
   } envia_tcp;

   struct recb_tcp
   {
      struct iphdr ip;
      struct tcphdr tcp;
      char buffer[10000];
   } recb_pct;


   /* Código original: synhose.c by knight */

   struct pseudo_header
   {
      unsigned int endereco_origem;
      unsigned int endereco_destino;
      unsigned char placeholder;
      unsigned char protocolo;
      unsigned short tcp_length;
      struct tcphdr tcp;
   } pseudo_header;



   int ch;
```

```
   int envia_socket;

   int recb_socket;

   struct sockaddr_in sin;

   FILE *input;

   FILE *output;



/* Inicializa o Gerador de Números Randômicos */

srand((getpid())*(porta_destino));



/**********************/

/* Código do Cliente  */

/**********************/


if(server==0)
{
if((input=fopen(filename,"rb"))== NULL)
 {
 printf("Impossivel abrir o arquivo %s para leitura\n",filename);
 exit(1);
 }

else while((ch=fgetc(input)) !=EOF)
 {

/* Ajusta tempo entre cada envio de pacote, o padrão é de 1 segundo*/
```

```
usleep(delay);


   /* Constrói o cabeçalho IP com as informações forjadas */
   envia_tcp.ip.ihl = 5;
   envia_tcp.ip.version = 4;
   envia_tcp.ip.tos = 0;
   envia_tcp.ip.tot_len = htons(40);

/* Gera um valor randômico para o campo de identificação do IP */

   envia_tcp.ip.id =(int)(255.0*rand()/(RAND_MAX+1.0));

   envia_tcp.ip.frag_off = 0;
   envia_tcp.ip.ttl = 64;
   envia_tcp.ip.protocol = IPPROTO_TCP;
   envia_tcp.ip.check = 0;
   envia_tcp.ip.saddr = ip_origem;
   envia_tcp.ip.daddr = ip_destino;


/* Constrói o cabeçalho TCP com as informações forjadas */

if(porta_origem == 0) /* gera porta de origem randomica caso nao seja
fornecida */

   envia_tcp.tcp.source = 1+(int)(10000.0*rand()/(RAND_MAX+1.0));

else
   envia_tcp.tcp.source = htons(porta_origem);


 /* Inseri a informação codificada no cabeçalho. */
```

```
   envia_tcp.tcp.seq = ch;
   envia_tcp.tcp.dest = htons(porta_destino);
   envia_tcp.tcp.ack_seq = 0;
   envia_tcp.tcp.res1 = 0;
   envia_tcp.tcp.doff = 5;
   envia_tcp.tcp.fin = 0;
   envia_tcp.tcp.syn = 1;
   envia_tcp.tcp.rst = 0;
   envia_tcp.tcp.psh = 0;
   envia_tcp.tcp.ack = 0;
   envia_tcp.tcp.urg = 0;
   envia_tcp.tcp.window = htons(512);
   envia_tcp.tcp.check = 0;
   envia_tcp.tcp.urg_ptr = 0;

   /* Entrega a informação forjada para a estrutura */

   sin.sin_family = AF_INET;
   sin.sin_port = envia_tcp.tcp.source;
   sin.sin_addr.s_addr = envia_tcp.ip.daddr;



   /* Abre socket cru para envio. */

   envia_socket = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
   if(envia_socket < 0)
   {
      perror("Impossivel enviar socket.\n Voce deve estar logado como
root para efetuar essa operação!");
      exit(1);
   }



      /* Faz o check sum do cabeçalho IP */
      envia_tcp.ip.check = in_cksum((unsigned short *)&envia_tcp.ip, 20);
```

```
      /* Preparação do cabeçalho por completo */



      /* From synhose.c by knight */
      pseudo_header.endereco_origem = envia_tcp.ip.saddr;
      pseudo_header.endereco_destino = envia_tcp.ip.daddr;
      pseudo_header.placeholder = 0;
      pseudo_header.protocolo = IPPROTO_TCP;
      pseudo_header.tcp_length = htons(20);


      bcopy((char *)&envia_tcp.tcp, (char *)&pseudo_header.tcp, 20);

      /* Ultimo checksum do pacote por completo */

      envia_tcp.tcp.check  =  in_cksum((unsigned  short  *)&pseudo_header,
32);

      /* Envio pacote */

      sendto(envia_socket,  &envia_tcp,  40,  0,  (struct  sockaddr  *)&sin,
sizeof(sin));

      printf("Enviando Dados: %c\n",ch);



  close(envia_socket);
 } /* fina do while(fgetc()) loop */
fclose(input);
}/* final do if(server == 0) loop */



/***********************/
/* Código do Servidor  */
/***********************/
```

```
else
{
 if((output=fopen(filename,"wb"))== NULL)
  {
  printf("Impossivel abrir o arquivo %s para escrita\n",filename);
  exit(1);
  }

/* Le o socket esperando as informações */

 while(1) /* loop de leitura do pacote */
 {
   /* Abre socket para leitura */
   recb_socket = socket(AF_INET, SOCK_RAW, 6);
   if(recb_socket < 0)
   {
      perror("Impossivel receber socket.\n Voce deve estar logado como
root para efetuar esta operacao");
      exit(1);
   }

  /* Escuta por um pacote de retorno num socket passivo */

  read(recb_socket, (struct recb_tcp *)&recb_pct, 9999);



        /* Se o pacote tiver a flag SYN/ACK setada e for do endereço
certo fazemos: */

        if (porta_origem == 0)
        {
                if((recb_pct.tcp.syn  ==  1)  &&  (recb_pct.ip.saddr  ==
ip_origem))
                {
                        /* Decode do ISN */
                        printf("Recebendo Dados: %c\n",recb_pct.tcp.seq);
                        fprintf(output,"%c",recb_pct.tcp.seq);
```

```
                        fflush(output);
                }
        }



/* Utiliza o pacote se o mesmo estiver destinado a porta serta e tiver as
flags de SYN/ACK setadas*/


        else
        {
                if((recb_pct.tcp.syn==1)  &&  (ntohs(recb_pct.tcp.dest)  ==
porta_origem))
                {
                        printf("Recebendo Dados: %c\n",recb_pct.tcp.seq);
                        fprintf(output,"%c",recb_pct.tcp.seq);
                        fflush(output);
                }
        }

   close(recb_socket); /* Fecha o socket*/
  }/* fim do loop do while() de leitura de pacotes */




  fclose(output);

 } /* Fin da função else(serverloop) */

} /* Fim da função pacotefalso() */



/* Função utilizada do algoritmo ping.c */

/* Copyright (c)1987 Regents of the University of California.
```

```
* All rights reserved.
*
* Redistribution and use in source and binary forms are permitted
* provided that the above copyright notice and this paragraph are
* dupliated in all such forms and that any documentation, advertising
* materials, and other materials related to such distribution and use
* acknowledge that the software was developed by the University of
* California, Berkeley. The name of the University may not be used
* to endorse or promote products derived from this software without
* specific prior written permission. THIS SOFTWARE IS PROVIDED ``AS
* IS'' AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
* WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHATIBILITY AND
* FITNESS FOR A PARTICULAR PURPOSE
*/

unsigned short in_cksum(unsigned short *ptr, int nbytes)
{
        register long             sum;              /* assumes long == 32
bits
*/
        u_short                  oddbyte;
        register u_short         answer;          /* assumes u_short == 16
bits */

        /*
         * Our algorithm is simple, using a 32-bit accumulator (sum),
         * we add sequential 16-bit words to it, and at the end, fold
back
         * all the carry bits from the top 16 bits into the lower 16
bits.
         */

        sum = 0;
        while (nbytes > 1)  {
                sum += *ptr++;
                nbytes -= 2;
        }
```

```
                                /* mop up an odd byte, if necessary */
        if (nbytes == 1) {
                oddbyte = 0;            /* make sure top half is zero */
                *((u_char *) &oddbyte) = *(u_char *)ptr;    /* one byte
only */
                sum += oddbyte;
        }

        /*
         * Add back carry outs from top 16 bits to low 16 bits.
         */

        sum  = (sum >> 16) + (sum & 0xffff);     /* add high-16 to low-16
*/
        sum += (sum >> 16);                     /* add carry */
        answer = ~sum;           /* ones-complement, then truncate to 16
bits
*/
        return(answer);
} /* end in_cksm()


/* Resolvedor de endereços, fonte desconhecida */
unsigned int host_convert(char *hostname)
{
   static struct in_addr i;
   struct hostent *h;
   i.s_addr = inet_addr(hostname);
   if(i.s_addr == -1)
   {
      h = gethostbyname(hostname);
      if(h == NULL)
      {
         fprintf(stderr, "Impossivel resolver %s\n", hostname);
         exit(0);
      }
      bcopy(h->h_addr, (char *)&i.s_addr, h->h_length);
   }
```

```
   return i.s_addr;
} /* end resolver */


/* Ajuda básica */
void ajuda(char *progname)
{
      printf("AJUDA\nStealth  TCP:  \n%s  -destino  ip_destino  -origem
ip_origem -arquivo nomedoarquivo -porta_origem porta -porta_destino porta
-server\n\n", progname);
      printf("-destino ip_destino       - Host para onde os dados serão
enviados.\n");
      printf("-origem ip_origem          - Host de onde os dados serão
ENVIADOS.\n");
      printf("                          No modo SERVIDOR este é o host de
onde os dados\n");
      printf("                          serão RECEBIDOS.\n");
      printf("-porta_origem porta       - Porta de onde os dados serão
originados. \n");
      printf("                                 (Gerada randomicamente por
padrão)\n");
      printf("-porta_destino porta    - Porta de destino dos dados.\n");
      printf("                            No modo SERVIDOR esta é a porta
onde os dados\n");
      printf("                          estarão sendo recebidos. Porta 80
por padrão.\n");
      printf("-arquivo nomedoarquivo  - Nome do arquivo para enviar.\n");
      printf("-delay tempo              - Tempo em microsegundos entre o
envio de cada\n");
      printf("                             Pacote. Para 1 segundo use: -
delay 1000000.\n");
      printf("-server                    - Modo servidor para aguardar o
recebimento de dados..\n");
      getchar();
      exit(0);
} /* fim da ajuda() */
/* The End */
```

9.2. Apêndice 2 – Código fonte do Menu

```
#!/bin/bash
# Exemplo Final de Script Shell
Principal() {
   clear
   echo "========================================="
   echo "UniCEUB - Projeto Final"
   echo "Stealth TCP"
   echo "Autor: Bruno Guedes Souto"
   echo "========================================="
   echo "OpÃ§Ãµes:"
   echo
   echo "1. Gerar Chave Privada"
   echo "2. Gerar Chave PÃºblica"
   echo "3. Gerar Mensagem Para Envio"
   echo "4. Cifrar Arquivo"
   echo "5. Decifrar Arquivo"
   echo "6. Enviar Arquivo Em Modo Stealth"
   echo "7. Modo Servidor Para Receber Arquivo"
   echo "8. Sair do Programa"
   echo
   echo -n "Qual a opÃ§Ã£o desejada? "
   read opcao
   case $opcao in
      1) GerarCPriv ;;
      2) GerarCPub ;;
      3) GerarMsg ;;
      4) Cifra ;;
      5) Decifra ;;
      6) Envia ;;
      7) ModoServer ;;
      8) exit ;;
      *) "OpÃ§Ã£o desconhecida." ; echo ; Principal ;;
   esac
}

GerarCPriv() {
   clear
   echo "Digite o nome do arquivo onde a chave privada serÃ¡ armazenada:
"
   read NomeFileCPriv

   if [ ${#NomeFileCPriv} -gt 15 ]; then
      echo "Nome de Arquivo muito grande."
      echo "Digite um nome de arquivo com menos de 15 caracteres."
      read
      GerarCPriv
   elif [ -f $NomeFileCPriv ]; then
      echo "Arquivo jÃ¡ existe, crie um arquivo com outro nome."
      read
      GerarCPriv
```

```
   else
      openssl genrsa -out "$NomeFileCPriv" 1024
      echo "Chave Privada gerada com sucesso e armazenada no arquivo
$NomeFileCPriv"
      read
   fi
   Principal
}

GerarCPub() {
   clear
   echo "Digite o nome do arquivo onde se encontra a chave privada: "
   read NomeFileCPriv

   if [ ! -f $NomeFileCPriv ]; then
      echo "Arquivo nÃ£o existe!"
      echo "Ã‰ preciso criar a chave publica a partir de um arquivo chave
privada existente."
      read
      GerarCPub
   elif [ -f "$NomeFileCPriv" ]; then
      echo "Digite o nome do arquivo onde a chave pÃºblica serÃ¡
armazenada: "
      read NomeFileCPub
      if [ ${#NomeFileCPub} -gt 15 ]; then
         echo "Nome de Arquivo muito grande."
         echo "Digite um nome de arquivo com menos de 15 caracteres."
         read
         GerarCPub
      elif [ -f $NomeFileCPub ]; then
         echo "Arquivo jÃ¡ existe, crie um arquivo com outro nome."
         GerarCPub
      else
         openssl rsa -in "$NomeFileCPriv" -pubout > "$NomeFileCPub"
         echo "Chave PÃºblica gerada com sucesso e armazenada no arquivo
$NomeFileCPub"
         read
      fi
   fi
   Principal
}

GerarMsg() {
   clear
   echo "Digite o nome do arquivo onde a mensagem serÃ¡ armazenada: "
   read NomeFileMsg

   if [ -f $NomeFileMsg ]; then
      echo "Arquivo jÃ¡ existe, crie um arquivo com outro nome."
      GerarCPub
   elif [ ${#NomeFileMsg} -gt 15 ]; then
      echo "Nome de Arquivo muito grande."
      echo "Digite um nome de arquivo com menos de 15 caracteres."
      read
      GerarCPub
   else
```

```
   echo "Digite a mensagem que serÃ¡ armazenada no arquivo $NomeFileMsg:
"
   read mensagem
   while [ ${#mensagem} -lt 10 ]; do
      echo "Mensagem muito pequena! Digite uma mensagem com mais de 10
caracteres."
      read mensagem
   done
      echo "$mensagem" > "$NomeFileMsg"
      echo "Mensagem armazenada no arquivo $NomeFileMsg"
      read
   fi
   Principal
}

Cifra() {
   clear
   echo "Digite o nome do arquivo onde se encontra a chave pÃºblica: "
   read NomeFileCPub

   if [ ! -f $NomeFileCPub ]; then
      echo "Arquivo nÃ£o existe!"
      echo "Ã‰ preciso apontar um arquivo de chave pÃºblica vÃ¡lido."
      read
      Cifra
   fi

   echo "Digite o nome do arquivo onde se encontra a mensagem: "
   read NomeFileMsgPub

   if [ ! -f $NomeFileMsgPub ]; then
      echo "Arquivo nÃ£o existe!"
      echo "Ã‰ preciso apontar um arquivo de mensagem vÃ¡lido."
      read
      Cifra
   fi

   echo "Digite o nome do arquivo onde irÃ¡ ser armazenado a mensagem
cifrada: "
   read NomeFileMsgPriv

   if [ -f $NomeFileMsgPriv ]; then
      echo "Arquivo jÃ¡ existe, crie um arquivo com outro nome."
      Cifra
   elif [ ${#NomeFileMsgPriv} -gt 15 ]; then
      echo "Nome de Arquivo muito grande."
      echo "Digite um nome de arquivo com menos de 15 caracteres."
      read
      Cifra
   else
      openssl rsautl -encrypt -in "$NomeFileMsgPub" -pubin -inkey
"$NomeFileCPub" -out "$NomeFileMsgPriv"
      echo "Mensagem cifrada com sucesso e armazenada no arquivo
$NomeFileMsgPriv"
      read
   fi
   Principal
```

```
}

Decifra() {
   clear
   echo "Digite o nome do arquivo onde se encontra a chave privada: "
   read NomeFileCPriv

   if [ ! -f $NomeFileCPriv ]; then
      echo "Arquivo nÃ£o existe!"
      echo "Ã‰ preciso apontar um arquivo de chave privada vÃ¡lido."
      read
      Decifra
   fi

   echo "Digite o nome do arquivo onde se encontra a mensagem cifrada: "
   read NomeFileMsgPriv

   if [ ! -f $NomeFileMsgPriv ]; then
      echo "Arquivo nÃ£o existe!"
      echo "Ã‰ preciso apontar um arquivo de mensagem cifrada vÃ¡lido."
      read
      Decifra
   fi

   echo "Digite o nome do arquivo onde irÃ¡ ser armazenado a mensagem
decifrada: "
   read NomeFileMsgPub

   if [ -f $NomeFileMsgPub ]; then
      echo "Arquivo jÃ¡ existe, crie um arquivo com outro nome."
      Decifra
   elif [ ${#NomeFileMsgPub} -gt 15 ]; then
      echo "Nome de Arquivo muito grande."
      echo "Digite um nome de arquivo com menos de 15 caracteres."
      read
      Decifra
   else
      openssl rsautl -decrypt -in $NomeFileMsgPriv -inkey $NomeFileCPriv
-out $NomeFileMsgPub
      echo "Mensagem decifrada com sucesso e armazenada no arquivo
$NomeFileMsgPriv"
      read
   fi
   Principal
}

Envia() {
   clear

   echo "Digite o nome do arquivo onde se encontra a mensagem cifrada: "
   read NomeFileMsgPriv

   echo "Digite o endereÃ§o IP de destino dos pacotes:"
   read IpDestino

   echo "Digite o endereÃ§o IP de origem dos pacotes:"
   read IpOrigem
```

```
   echo "Digite a porta de Origem dos Pacotes:"
   read PortaOrigem

   echo "Digite a porta de Destino dos Pacotes:"
   read PortaDestino

   echo "Digite o Delay de envio entre os pacotes em micro segundos (1
segundo = 1000000):"
   read DelayPacote

   ./stealth_tcp -origem $IpOrigem -destino $IpDestino -arquivo
$NomeFileMsgPriv -porta_origem $PortaOrigem -porta_destino $PortaDestino
-delay $DelayPacote

   read

   Principal
}

ModoServer() {
   clear

   echo "Digite o nome do arquivo onde serÃ¡ salvo a mensagem cifrada: "
   read NomeFileMsgPriv

   echo "Digite o endereÃ§o IP de origem dos pacotes:"
   read IpOrigem

   echo "Digite a porta de destino dos Pacotes:"
   read PortaOrigem

   ./stealth_tcp -origem $IpOrigem -arquivo $NomeFileMsgPriv -
porta_origem $PortaOrigem -server

   read
   Principal
}

Principal
```